TEKNISKA HÖGSKOLAN

Elektrotekniska avdelningen

Sverre Slotte

DRIFTSTÖDSPROTOKOLL FÖR MODEMSTATIV

Diplomarbete som inlämnats för granskning som lärdomsprov för avläggande av diplomingenjörsexamen i Esbo den 16.4.1992.

Arbetets övervakare Reijo Sulonen

Arbetets handledare Ari Ahtiainen

Utfört av: Sverre Slotte

Arbetets namn: Driftstödsprotokoll för modemstativ

Datum: 16.4.1992 **Sidantal:** 70

Avdelning: Elektrotekniska avdelningen **Professur:** Tik-76 Programvaruteknik

Övervakare: Professor Reijo Sulonen

Handledare: Forskningschef Ari Ahtiainen

Diplomarbetet bestod i att portera en protokollstack för driftstöd till styrkortet i ett modemstativ, så att man från en arbetstation skall kunna utföra driftstödsoperationer på de i stativet befintliga modemen.

Kommunikationen mellan modemstativet och arbetsstationen baserar sig på den protokollstack som är definierad i den Internationella standardiseringsorganisationens referensmodell för öppna system (OSI-modellen).

Driftstöd i enlighet med referensmodellen för öppna system är en distribuerad databehandlingstillämpning som använder standardiserade protokoll för kommunikation. Driftstöd behandlas kortfattat i diplomarbetets teoretiska del.

Protokollstackens uppgift är att erbjuda driftstödstjänster åt driftstödstillämpningen. I protokollstackens tillämpningsskikt finns därför ett tillämpningselement för driftstöd (CMISE). Kommunikationen mellan de samverkande enheterna sker i enlighet med ett driftstödsprotokoll (CMIP).

Protokollstacken utvecklades med hjälp av ett vid Statens tekniska forskningscentral framtaget programmeringsverktyg för kommunikationsprotokoll (CVOPS). För implementering av ASN.1 baserade protokoll användes en vid Nokia Forskningscentral utvecklad kompilator (CASN).

På modemstativets styrkort finns ett leverantörsspecifikt operativsystem (DNOS). Utvecklingsmiljön anpassades till operativsystemet, varvid speciell vikt lades på den asynkrona kommunikationen mellan processerna och omgivningen. Förutom själva protokollstacken gjordes även några stödprocesser och en rudimentär testapplikation.

Nyckelord: Driftstöd, nätövervakning, OSI

Author: Sverre Slotte

Name of the thesis: Network management protocol for a modem rack

Date: April 16, 1992 **Number of pages:** 70

Department: Faculty of Electrical Engineering **Professorship:** Software Technology

Supervisor: Professor Reijo Sulonen

Instructor: R&D Manager Ari Ahtiainen

This work consisted of porting a network management protocol stack to the control card of a modem rack in order to enable manipulation of the modems in the rack from a workstation.

The communication between the modem rack and the workstation is based on the protocol stack defined by the International Organisation for Standardization in the reference model for open systems (the OSI model).

Network management according to the reference model for open systems is a distributed data processing application. Network management is briefly discussed in the theoretical part of the thesis.

The role of the protocol stack is to offer network management services to the application. In order to do so the application layer of the protocol stack contains an application service element for network management (CMISE). The two cooperating entities communicate by means of a network management protocol (CMIP).

The protocol stack was developed using a programming tool (CVOPS) from the Technical Research Centre of Finland. ASN.1-based protocols were implemented using a compiler developed by Nokia Research Center (CASN).

The control card of the modem rack hosts a vendor specific operating system (DNOS), to which the developement environment was ported. The asynchronous communication between the processes and the environment were given special attention. In addition to the protocol stack some support processes and a rudimentary test application were also implemented.

Keywords: Network management, OSI

Förord

Detta diplomarbete gjordes vid Nokia Forskningscentral på avdelningen för signalering och protokoll. Jag vill tacka Nokia Forskningscentral och Nokia Datakommunikation som gjorde detta diplomarbete möjligt.

Som diplomarbetets övervakare verkade professor Reijo Sulonen på Tekniska högskolan och som handledare forskningschef Ari Ahtiainen på Nokia Forskningscentral. Till dem utsträcker jag ett varmt tack för det stöd de gett mig.

Mikko J. Salminen på Nokia Datakommunikation förtjänar en eloge för att han tog initiativ till projektet och tålmodigt stödde det trots överraskande och omväxlande yttre betingelser.

Pekka Rouvinen, även han på Nokia Datakommunikation, rätade ut flera frågetecken kring DNOS:

```
tmsg.task = SVERRE;
tmsg.type = THANKYOU_MSG;
tmsg.msg  = "Tack för hjälpen!";

Send_msg (ROUVINEN, &tmsg);
```

Esbo 16.4.1992

Sverre Slotte

Använda förkortningar

ACSE	Association Control Service Element. Tillämpningselementet för samband i OSI-modellen.
AE	Application Entity, tillämpningsenhet. Den mängd OSI kommunikationsfunktioner som hänför sig till en viss tillämpningsprocess.
AP	Application Process, tillämpningsprocess. En tillämpningsprocess i OSI-modellen.
ASE	Application Service Element, tillämpningselement. En samling funktioner som tillhandahåller allmänna kommunikationstjänster för en viss form av samarbete mellan tillämpningsenheter.
ASN.1	Abstract Syntax Notation One. Ett språk för beskrivning av datastrukturers logiska uppbyggnad.
BER	Basic Encoding Rules. Kodningsregler med vilka ASN.1-datastrukturerna kan kodas på ett maskinoberoende sätt.
CASN	Compiler for ASN.1. En kompilator för ASN.1 som utvecklats vid Nokia Forskningscentral.
CCITT	Comité Consultatif Internationale Télégraphique et Téléphonique. En internationell samarbetsorganisation för nationella post- och televerk.
CEP	Connection End Point, förbindelseändpunkt.
CMIP	Common Management Information Protocol. Protokollet för driftstöd i OSI-modellen.
CMISE	Common Management Information Service Element. Tillämpningselementet för driftstöd i OSI-modellen.
CVOPS	C language Virtual Operating System. En vid Statens tekniska forskningscentral utvecklad protokollutvecklingsomgivning.
EFSA	Extended Final State Automaton. En utvcklad, ändlig tillståndsautomat.
HDLC	High Level Data Link Control. Ett kommunikationsprotokoll för länkskiktet i OSI-modellen.
IAB	Internet Activities Board. Organisation som leder utvecklingsarbetet av Internet-protokoll.
ISO	Internationella standardiseringsorganisationen.

LAPB	Link Access Procedure, Balanced. Ett kommunikationsprotokoll för länkskiktet i OSI-modellen.
OSI	Open Systems Interconnected. En av ISO standardiserad referensmodell för kommunikation mellan öppna system.
PCI	Protocol Control Information, styrinformation för protokoll.
PDU	Protocol Data Unit, protokolldataenhet.
ROSE	Remote Operation Service Element. Tillämpningselementet för fjärroperationer i OSI-modellen.
SAP	Service Access Point, punkt för tjänsteåtkomst.
SDU	Service Data Unit, tjänstedataenhet.

Innehållsförteckning

1. Inledning

I vårt samhälle har data- och telekommunikation redan länge spelat en mer och mer betydande roll. De allt större och interkopplade heterogena nätverk som uppstår kräver en väl genomtänkt underhållsstrategi för att kunna utnyttjas optimalt. Till följd av de stora ekonomiska intressen som står på spel måste eventuella fel snabbt kunna rättas till, och om en del av nätet av en eller annan orsak försätts ur funktion måste en omkonfigurering kunna ske utan dröjsmål. Driftstödet blir av vital betydelse.

Såväl internationella standardiseringsorganisationer som stora företag inom databranschen har definierat olika driftstödskoncept. Utmärkande för standardiseringsgremiernas rekommendationer är att de rör sig på ett tämligen abstrakt och naturligtvis leverantörsoberoende plan. De beskriver snarare generella metoder för driftstöd än specifika tillvägagångssätt.

I detta diplomarbete implementerades en sk. Q3-protokollstack för driftstöd i ett modemstativ. Arbetet gjordes i en UNIX arbetsstation i en emulator för modemstativets operativsystem DNOS. Kapitlen två till och med fem utgör en beskrivning av diplomarbetets ämnesområde. I dem beskrivs den nödvändiga bakgrunden och de använda verktygen. Kapitlen sex till och med nio befattar sig med diplomarbetets praktiska utförande.

I kapitel två i detta diplomarbete beskrivs den av den internationella standardiseringsorganisationen ISO standardiserade referensmodellen för kommunikation mellan öppna system (den sk. OSI-modellen). I det därpåföljande kapitlet kastar vi en blick på begreppet driftstöd.

OSI-protokollstacken är realiserad med hjälp av det vid Statens tekniska forskningscentral (VTT) utvecklade protokollutvecklingsverktyget CVOPS och den vid Nokia Forskningscentral (NRC) framtagna kompilatorn för ASN.1 CASN. Kapitel fyra belyser programutveckling med såväl CVOPS som CASN.

Modemstativets operativsystem DNOS presenteras kort i kapitel fem.

I kapitel sex beskrivs själva arbetets utförande, dvs. den utvecklade kommunikationsprogramvarans arkitektur och de individuella processernas roll. Där redogörs också för den använda apparaturen.

Kapitel sju behandlar porteringen av CVOPS till DNOS. I kapitel åtta och nio beskrivs protokollstackens bägge gränssnittuppdrag.

2. OSI referensmodell

Datorsystem som kan kommunicera enbart med utrustning från samma leverantör kallas *slutna system*. I slutna system kan informationsteknologins möjligheter vad t.ex. nätverk beträffar inte utnyttjas på ett kostnadseffektivt sätt. Man blir tvungen att för ett slutet system bygga en speciell övergångsmodul för varje annan sluten nätverksarkitektur man vill kommunicera med. Vissa prominenta firmor i databranschen, som icke sällan kärnfullt förkortar sitt namn till tre bokstäver, levererar kommunikationslösningar som anses höra till kategorin slutna system.

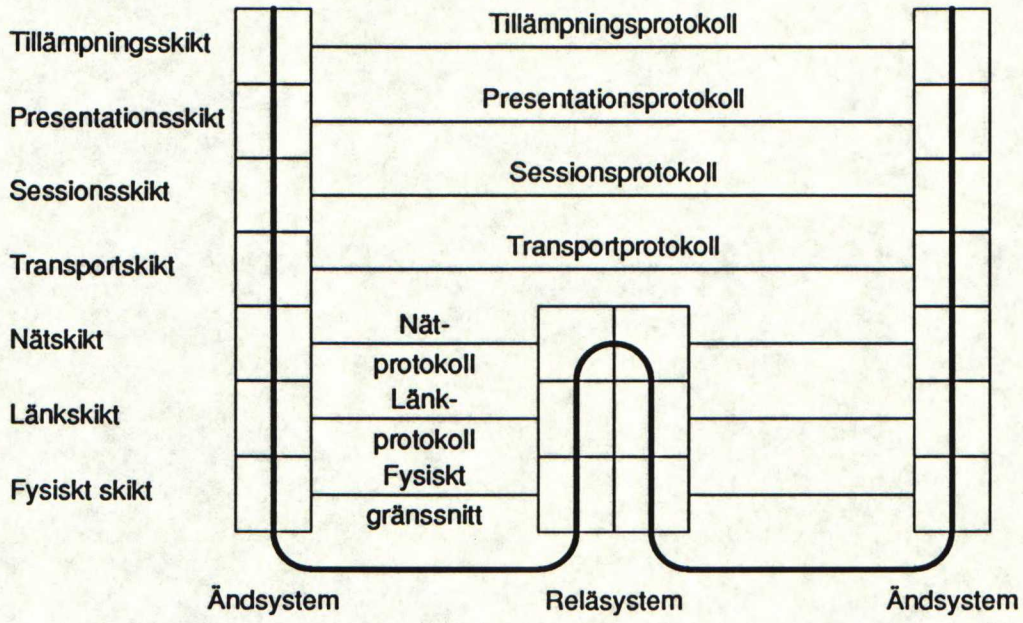
För att avhjälpa dessa problem har den internationella standardiseringsorganisationen ISO utvecklat en referensmodell för kommunikation mellan sk. *öppna system*. Modellen kallas *Open Systems Interconnection*, förkortat OSI [ISO 1984]. Ett öppet system kan samarbeta med andra öppna system som tillämpar OSI-standarder.

OSI referensmodellen beskriver en abstrakt struktur för en skiktad indelning av kommunikationsfunktioner. Den innehåller inga standarder eller rekommendationer för hur funktionerna skall utföras i praktiken, utan utgör en ram för samordning av befintliga och kommande standarder. ISO kallar modellen *the ISO Seven Layer, Open Systems Interconnection, Basic Reference Model*.

2.1 En översikt av referensmodellen

Den grundläggande tanken i OSI är att man kan dela upp en godtycklig datakommunikationsprocess i ett antal funktionellt avgränsade och någorlunda väldefinierade steg [Abramowicz 1989]. Ett öppet system indelas i OSI-modellen i sju separata skikt (se figur 1). De fyra understa skikten ansvarar för transporten av data mellan kommunikationspartnerna. De två därpåföljande skikten erbjuder mer tillämpningsnära tjänster, medan det sjunde skiktet tillhandahåller direkt applikationsspecifikt stöd för kommunikationen. Tillämpningen innefattas inte i OSI referensmodellen.

Ett skikt erbjuder vissa tjänster åt skiktet närmast ovanför, även kallat användaren av tjänsterna. För att göra detta använder skiktet dels funktioner inom skiktet självt, dels tjänster som tillhandahålls av skiktet nedanför. Man kan säga, att ett skikt förädlar och utvidgar de tjänster som det underliggande skiktet tillhandahåller.



Figur 1. OSI referensmodell [Abramowicz 1989]

OSI standarder innehåller definitioner för de funktioner som skall finnas inom varje skikt och de tjänster skiktet skall utföra åt närmast högre skikt. Tjänstedefinitionen ligger sedan till grund för specificeringen av det protokoll med vars hjälp funktionsenheterna inom ett skikt i ett system samverkar. För varje skikt skall således finnas en standard som definierar *tillgängliga tjänster* och dessutom specifikationer för en eller flera *protokoll*standarder. En samling skiktade protokoll kallas ofta för en *protokollstapel* eller *protokollstack.*

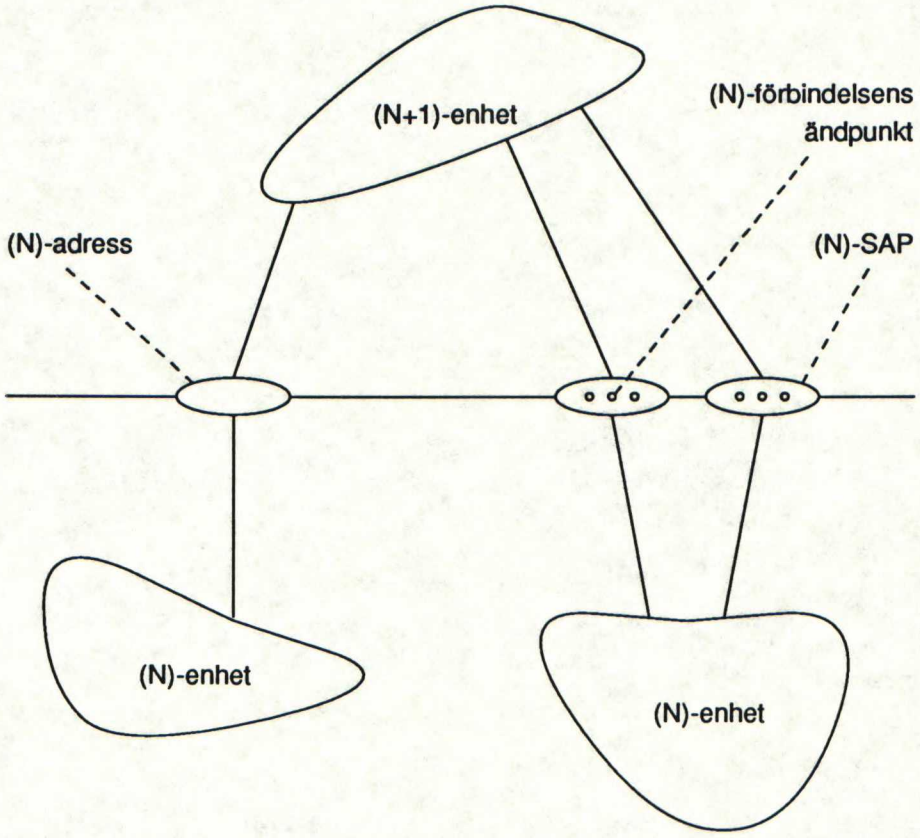
Den väldefinierade, modulära skiktstruktur som OSI-modellen uppvisar innebär att ett protokoll inom ett skikt kan bytas ut utan att protokollstapelns funktion påverkas. Det enda som krävs är att skiktet fortfarande kan erbjuda samma tjänster åt användaren som tidigare.

Datautbytet mellan två kommunicerande enheter kan vara antingen *förbindelseinriktat* (connection oriented) eller *förbindelsefritt* (connectionless mode). En förbindelseinriktad kommunikationsprocess omfattar tre faser: upprättande av förbindelsen, dataöverföring och brytande av förbindelsen. Då förbindelsen väl är etablerad kan stora datamängder snabbt och effektivt transporteras.

Om behovet att överföra data är sporadiskt återkommande och mängderna mer blygsamma kan förbindelsefri överföring vara att rekommendera. Överföringen sker snabbare, men avsändaren kan inte vara säker på att meddelandet når adressaten. Förbindelsefri överföring är vanlig i lokalnät (LAN).

2.2 Ett generiskt skikt

Den aktiva komponenten i ett skikt i OSI referensmodellen kallas *skiktenhet*. Skiktenheten i ett godtyckligt skikt (N) tillhandahåller tjänster som utnyttjas av enheten i skikt (N+1). Dessa tjänster kallas (N)-tjänster. Skikt (N) begagnar sig i sin tur av (N-1)-tjänster. Gränssnittet mellan två skikt innehåller en *punkt för tjänsteåtkomst* (SAP, Service Access Point). Detta är illustrerat i figur 2. En tjänsteåtkomstpunkt får sitt namn efter det skikt som erbjuder tjänsterna. Sålunda kallas åtkomstpunkten för sessionstjänster S-SAP (Session Service Access Point). Den ligger alltså på gränsen mellan sessions- och presentationsskiktet.

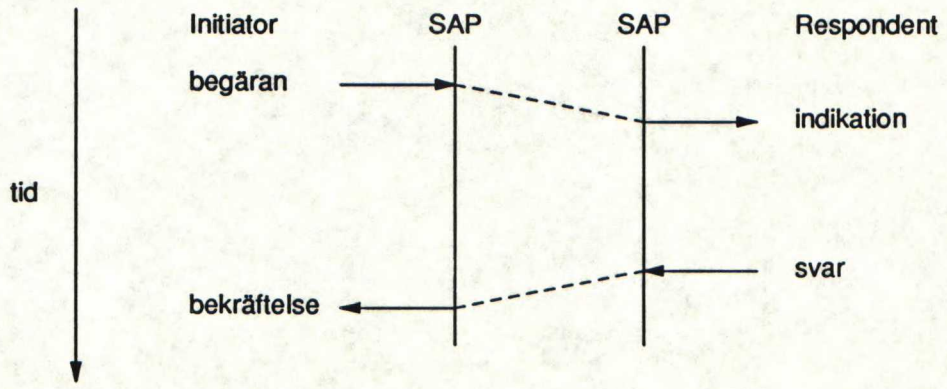


Figur 2. Tjänsteåtkomstpunkt och förbindelseändpunkt [ISO 1984]

En skiktenhet kan samtidigt hantera flera parallella förbindelser. Inom skiktets tjänsteåtkomstpunkt finns då för varje förbindelse en *förbindelseändpunkt* (CEP, Connection End Point).

De tjänster ett skikt skall tillhandahålla eller erbjuda åt det överliggande skiktet beskrivs i form av elementära tjänster, sk. *tjänsteprimitiv* (service primitives), av vilka det finns fyra grundtyper: *begäran* (request), *indikering* (indication), *svar* (response) och

bekräftelse (confirmation). Varje primitiv kan innefatta valbara parametrar som påverkar kommunikationsprocessen. Ett komplett primitivnamn, t.ex. S-DISCONNECT.request, består av en bokstav som anger skikt (S), ett verb som anger operation (DISCONNECT) och till sist en grundtyp (request).

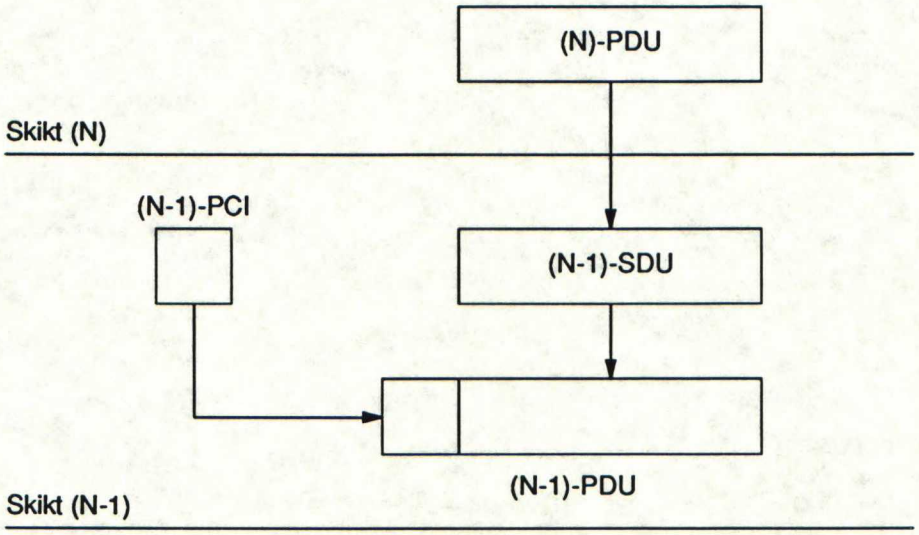


Figur 3. Tjänsteprimitivens kronologiska samband [Rose 1990 s. 21]

Initiatorn eller användaren av en tjänst sänder en begäran till skiktenheten nedanför. Respondenten får en indikation genom sin tjänsteåtkomstpunkt. I icke-bekräftade tjänster används endast dessa två grundtyper. Om tjänsten måste bestyrkas sänder respondenten ett svar till sitt underliggande skikt. Svaret når initiatorn i form av en bekräftelse. Detta förlopp är illustrerat i figur 3.

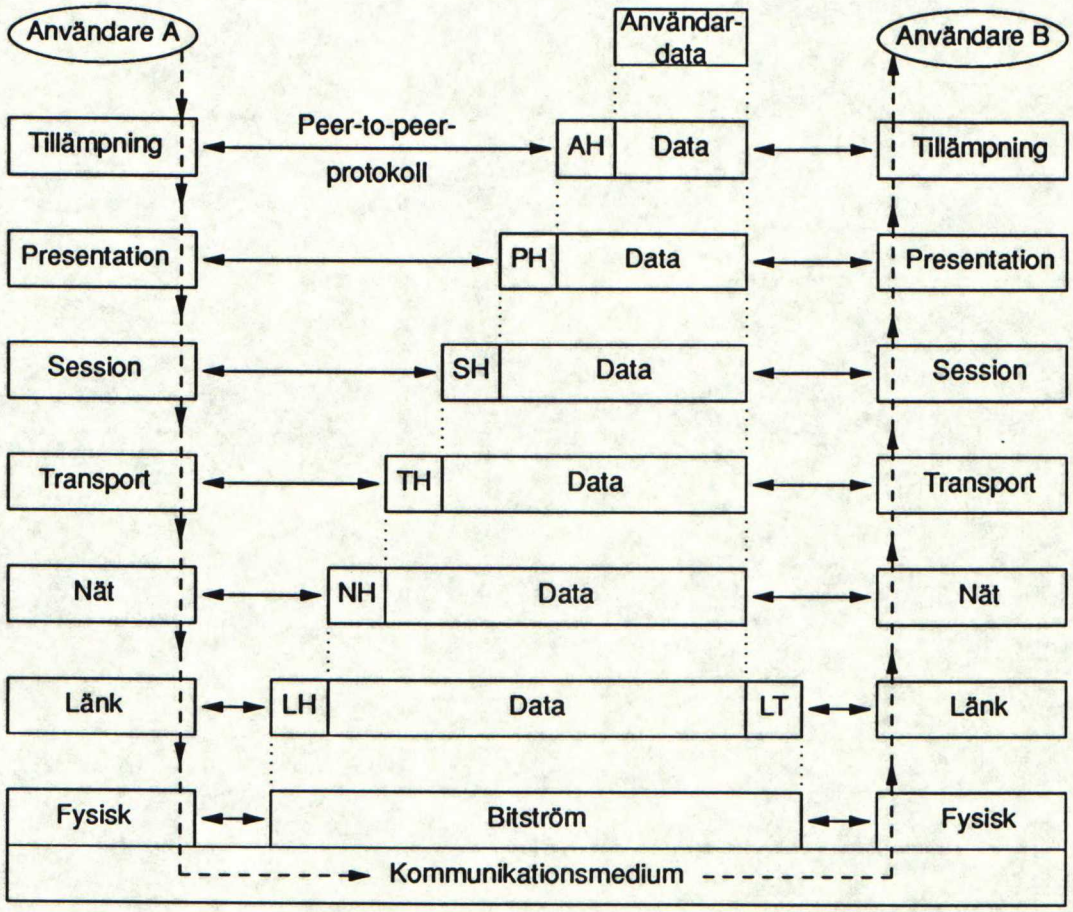
Två med varandra kommunicerande, i olika system belägna skiktenheter kallas *partnerenheter*. Partnerenheterna i skikt (N) kommunicerar med varandra med hjälp av ett (N)-protokoll. Partnerenheterna sänder *protokolldataenheter* (PDU, Protocol Data Unit) till varandra.

För att överföra en (N)-PDU placeras den som dataparameter i ett (N-1)-tjänsteprimitiv. Tjänsteprimitivet kan innehålla också andra parametrar, t.ex. initiatorns adress. På detta vis bildas en *(N-1)-tjänstedataenhet* (SDU, Service Data Unit). Skikt (N-1) bildar en (N-1)-PDU genom att till denna (N-1)-SDU lägga ett fält med *styrinformation för (N-1)-protokoll* (PCI, Protocol Control Information). Detta är beskrivet i figur 4.



Figur 4. Sambandet mellan SDU och PDU [ISO 1984]

På detta vis rör sig dataenheten nedåt i protokollstacken som en hiss. Varje skikt lägger till sin egen kontrollinformation. Det fysiska skiktet överför datablocket via ett kommunikationsmedium till mottagarstacken. Där åker dataenheten uppåt, varvid varje skikt skalar av sin egen kontrollinformation. Till sist får användarprocessen ta emot en tjänsteindikation innahållande dataenheten. Se figur 5.



Figur 5. Informationsflödet i protokollstacken [Abramowicz 1989]

2.3 Skikten 1 - 6

2.3.1 Det fysiska skiktet — Physical Layer

Det fysiska skiktets uppgift är att förmedla en bitström mellan två länkskikt. Standarderna för det fysiska skiktet definierar bl.a. överföringsrat, kontaktdonens mekaniska dimensioner och spänningsnivåer.

2.3.2 Länkskiktet — Data Link Layer

Länkskiktet tillhandahåller en, i princip, felfri punkt-till-punkt förbindelse mellan nätskikt. Skiktet har resurser för att upptäcka fel, omsändning och flödesstyrning. En länkförbindelse består av en eller flera fysiska förbindelser.

2.3.3 Nätskiktet — Network Layer

Nätskiktets grundläggande uppgift är att tillhandahålla en transparent kanal för överföring av data mellan transportskikt i skilda system. Den dirigerar vid behov kanalen genom flera mellanliggande reläsystem. Skiktet etablerar, upprätthåller och avslutar förbindelser mellan systemen och hanterar adresseringen. Flera förbindelser mellan transportskikt kan multiplexeras på en nätförbindelse.

2.3.4 Transportskiktet — Transport Layer

Transportskiktet fungerar som isolering av de övre skikten så att dessa blir oberoende av underliggande näts egenskaper. Transportskiktet erbjuder en direkt förbindelse av tillräcklig kvalitet mellan de kommunicerande ändsystemen. Tjänstekvaliteten bestäms av parametrar vid tiden för etablerandet av förbindelsen.

Beroende på de underliggande bärarnätens kvalitet kan en av fem standardiserade tjänsteklasser, klass 0 till klass 4, väljas för en förbindelse. Klass 0 är tillräcklig då det underliggande bärarnätet har god kvalitet medan klass 4 används för att åstadkomma en kvalitetsförbättring när det använda nätets kvalitet är mindre god.

Transportskiktet kan optimera datakommunikationen genom att multiplexera eller spjälka upp dataflöden innan de når nätförbindelserna.

2.3.5 Sessionsskiktet — Session Layer

Sessionsskiktet etablerar och bryter förbindelser mellan presentationsskikt i skilda system och erbjuder styrning av dialogen dem emellan. I samband med upprättandet av förbindelse kommer samarbetande presentationsskikt överens om vilka funktioner som

önskas av sessionsskiktet.

En tjänst i skiktet understöder dialogen genom att tilldela parterna sändrättigheter. En annan tjänst tillåter presentationsskiktet att bestämma synkronisationspunkter i dataflödet från vilka dialogen kan återupptas efter störningar i dataöverföringen. Sammanlagt tillhandahåller sessionsskiktet ett tiotal valbara sk. *funktionella enheter* (functional units).

2.3.6 Presentationsskiktet — Presentation Layer

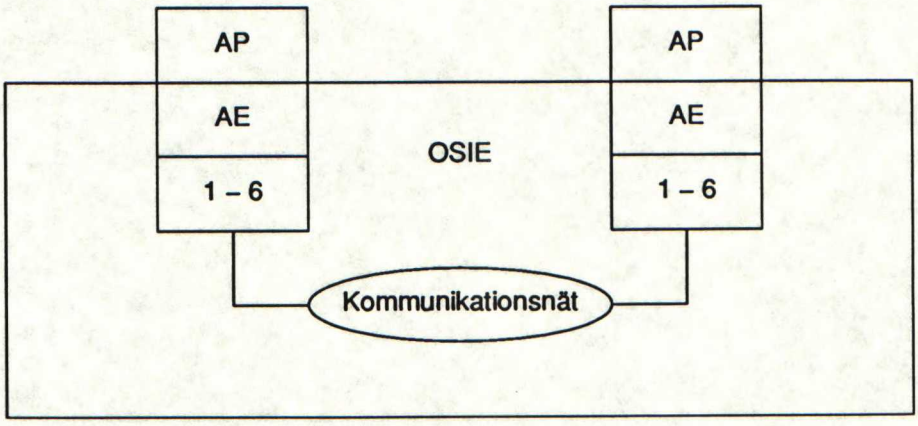
Presentationsskiktet definierar hur data syntaxmässigt skall representeras vid överföring. Den semantiska tolkningen överlämnas åt tillämpningsskiktet. De kommunicerande tillämpningarna har en gemensam abstrakt syntax. Presentationsskiktet transformerar den till en överföringssyntax som väljs i samband med upprättandet av en förbindelse.

2.4 Tillämpningsskiktet

Syftet med OSI standarder är att erbjuda kommunikationstjänster åt användnings- eller applikationsprogram som kräver koordinering av aktiveteter i två eller flera öppna system. Standarderna för tillämpningsskiktet definierar generella principer för stöd av distribuerad informationsbehandling [ISO 1989a]. Tillämpningsskiktet tillhandahåller tjänster för att stöda slutanvändarens användningsprocess och för att styra all kommunikation mellan applikationerna.

I OSI referensmodellen beskrivs kommunikationen mellan öppna system som samverkan mellan *tillämpningsprocesser* (Application Process, AP) i dessa system. En tillämpningsprocess representerar de resurser som utför själva informationsbehandlingen; den befattar sig på intet sätt med kommunikationsuppgifter.

En *tillämpningsenhet* (Application Entity, AE) representerar en mängd OSI kommunikationsfunktioner som hänför sig till en viss tillämpningsprocess. Den utgör med andra ord den del av applikationen som förser tillämpningsprocessen med kommunikationsmöjligheter. Tillämpningsenheten tillgodogör sig i sin tur den underliggande protokollstackens tjänster via tjänsteåtkomstpunkten P-SAP; tillsammans bildar de en sk. *OSI-omgivning* (OSI-environment). Figur 6 belyser sambandet mellan de nyss presenterade begreppen.



Figur 6. Tillämpningsprocesser i OSI-omgivningen [Kurronen 1991]

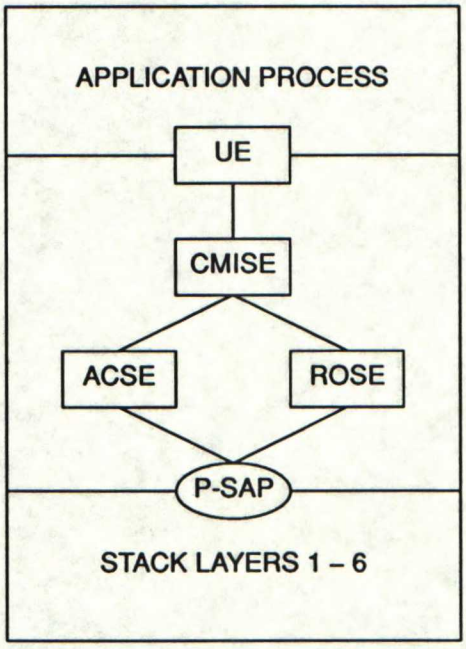
För att bringa klarhet och reda i tillämpningsskiktets struktur har man grupperat leverantörer av ofta använda kommunikationstjänster i *tillämpningselement* (ASE, Application Service Element). Ett tillämpningselement är en samling funktioner som tillhandahåller allmänna kommunikationstjänster för en viss form av samarbete mellan tillämpningsenheter. Tillämpningselement av en viss typ kommunicerar endast med andra tillämpningselement av samma typ. Kommunikationen mellan två tillämpningsenheter består alltså av flera underprotokoll; ett för varje typ av involverat tillämpningselement. Ett tillämpningselement kommunicerar med sina partnerenheter genom att utbyta *dataenheter för tillämpningsprotokoll* (APDU, Application Protocol Data Unit). Ett tillämpningselement kan utnyttja de tjänster ett annat tillämpningselement erbjuder.

För att informationsutbytet mellan två samverkande tillämpningsenheter skall vara möjligt måste de ha en gemensam regel- och kunskapsbas som styr kommunikationen. Denna bas utgör en *tillämpningskontext* (application context). Den innefattar regler för bl.a. vilka tillämpningselement som ingår i kommunikationen, hur gemensamma objekt identifieras och hur tillämpningskontexten kan modifieras. En tillämpningskontext kunde populärvetenskapligt sett anses vara en specifik samling tillämplingsprotokoll som understöder en viss tillämpningsprocess. Begreppet tillämpningskontext kan alltså i det närmaste jämföras med benämningen protokoll i de underliggande skikten.

Det tillämpningselement som ansvarar för etablering och brytning av förbindelser kallas *tillämpningselementet för styrning av samband*, populärt kallat ACSE (Association Control Service Element). Alla tillämpningselement i en tillämpningskontext begagnar sig av en gemensam förbindelse till partnerenheten som ACSE bygger upp och avslutar. Denna förbindelse skiljer sig från de underliggande förbindelserna i protokollstacken såtillvida, att flera protokoll använder sig av den. Därför har den fått ett speciellt namn; den kallas för *tillämpningssamband* (application association).

Figur 7 illustrerar tillämpningsskiktets interna struktur i en driftstödsapplikation. Boxen

märkt UE (User Element) utgör det sk. *användningselementet*. Det koordinerar tjänsteelementens samarbete och levererar tillämpningskontextens tjänster åt tillämpningsprocessen. Tillämpningselementen ROSE och CMISE behandlas mera ingående i för ändamålet dedikerade kapitel.



Figur 7. Tillämpningsskiktets interna struktur

Ett tillämpningssamband betjänar alltid en tillämpningskontext. Ingenting hindrar dock en tillämpningskontext från att samtidigt omfatta flera samband med flera tillämpningskontexter.

2.5 Abstrakt syntax, överföringssyntax och ASN.1

Tillämpningsprocesser samverkar genom att utväxla meddelanden. För att två tillämpningsprocesser skall kunna kommunicera med varandra krävs det att de har en gemensam begreppsvärld såväl semantiskt som syntatiskt. OSI referensmodellen, som standardiserar datautbytet mellan öppna system, befattar sig inte med applikationernas semantik. Syntaxen har däremot varit föremål för grundlig standardisering.

I datakommunikationens barndom specificerade man meddelandets uppbyggnad på bit-nivå. På detta vis kunde man undvika problem med t.ex. olika byteordning och

ordlängd. För att möjliggöra en precisare beskrivning av de allt mer komplexa datatyper som ingår i specifikationerna av moderna kommunikationsprotokoll har man på senare tid tagit formella definitionsspråk till hjälp.

En *abstrakt syntax* (abstract syntax) är den mängd formellt beskrivna datatyper av vilka meddelandena i ett kommunikationsprotokoll är uppbyggda. En gemensam abstrakt syntax är med andra ord en förutsättning för att två applikationer skall kunna kommunicera med varandra. Den abstrakta syntaxen kan beskrivas på ett naturligt språk, på ett programmeringsspråk som Pascal eller C eller på något formellt specifikationsspråk.

En abstrakt syntax kunde t.ex. definieras som en mängd av taltripletter som består av icke negativa heltal mindre än tiotusen. Till denna mängd hör bl.a. (0,0,7), (804,14,49) och (43,76,208).

För att uppmuntra protokollplanerare att begagna sig av ett enhetligt specifikationsspråk har ISO skapat en formell notation för abstrakta datatyper: ASN.1 (Abstract Syntax Notation One) [ISO 1987a]. ASN.1 definierar en samling enkla, grundläggade datatyper samt grammatikaliska regler för hur man utgående från dessa kan konstruera nya datatyper som är specifika för den aktuella tillämpningen. Protokollets abstrakta syntax fås genom att beskriva protokolldataenheterna med ASN.1.

Den ovan nämnda abstrakta syntaxen kunde ges namnet `Taltriplett` och i ASN.1 beskrivas på följande sätt:

```
-- tre icke negativa heltal mindre än tiotusen
Taltriplett ::= SEQUENCE {
                    forsta INTEGER ( 0..9999 ),
                    andra  INTEGER ( 0..9999 ),
                    tredje INTEGER ( 0..9999 )
                }
```

Kommentarer i ASN.1 börjar med två streck (--) och sträcker sig antingen till radens slut eller till nästa förekomst av två streck.

De i ASN.1 beskrivna datatyperna måste kunna kompileras till en maskinspecifik representation som kallas *konkret syntax* (concrete syntax) för att en tillämpningsenhet eller ett tillämpningselement skall kunna behandla dem. Datamaskiner av olika arkitektur har naturligtvis olika konkreta syntaxer, men ingenting hindrar att man för en och samma arkitektur definerar flera konkreta syntaxer. En sådan lokal variant kallas *lokal syntax* (local syntax).

En konkret syntax för exemplet med taltrippeln kunde vara t.ex. en datastruktur i C av arten `struct` bestående av tre heltal eller en ASCII teckensträng av typ "(0,0,7)".

För att två applikationer skall kunna kommunicera sinsemellan bör de utväxlade dataenheterna ha en gemensam konkret syntax, en sk. *överföringssyntax* (transfer syntax). Transformationen från lokal syntax till överföringssyntax sker i presentationsskiktet.

Den abstrakta syntaxen och överföringssyntaxen bildar tillsammans en *presentationskontext* (presentation context).

Det står var och en fritt att definiera sin egen överföringssyntax. ISO har standardiserat en universell överföringssyntax, som man erhåller genom att tillämpa speciella kodningsregler på de i ASN.1 definierade datatyperna. Dessa kodningsregler kallas BER (Basic Encoding Rules) [ISO 1987b].

BER är en oktettorienterad, flexibel uppsättning regler som erbjuder flera sätt på vilka en viss datatyp kan kodas. Varje kodning är dock entydig och innefattar förutom värdet också det kodade värdets typdefinition i ASN.1.

Om värdet (0,0,7) från vårt exempel kodas med BER kan vi få följande hexadecimala oktettsträng:

```
30 09 02 01 00 02 01 00 02 01 07
```

ASN.1 och BER används, förutom vid specifikation av OSI protokoll, också inom andra protokollfamiljer, t.ex. MAP/TOP, GSM och TCP/IP [Ahtiainen 1990 s. 347].

2.5.1 Tillämpningselementet för samband — ACSE

ACSE (Association Control Service Element) [ISO 1988a, 1988b] är det tillämpningselement som styr tillämpningssambandet mellan två tillämpningsenheter. Ett tillämpningssamband måste skapas innan någon som helst dataöverföring kan ske. Som en följd av detta ingår ACSE i alla OSI tillämpningar. ACSE befattar sig enbart med kontroll av samband och tillhandahåller bara tre tjänster: etablering, avveckling och brytning av samband. ACSE utnyttjar presentationsskiktets tjänster.

Etablering av tillämpningssamband är som helhet betraktat en ganska komplicerad process. Största delen av parametrarna, av vilka det finns över 20 stycken, passerar dock oförändrade rakt igenom till de undre protokollskikten. Att etablera ett tillämpningsssamband sker genom den bekräftade tjänsten A-ASSOCIATE.

Först etableras transportförbindelsen mellan partnerenheterna, varefter respondenten får tjänsteprimitivet A-ASSOCIATE.indication. En av parametrarna i respondentens svar (A-ASSOCIATE.response) indikerar huruvida respondenten accepterar eller förkastar begäran om etablering av samband. Denna parameter ingår naturligtvis också i bekräftelsen (A-ASSOCIATE.confirmation). Ifall den är positiv är såväl sessions- och presentationsförbindelsen som sambandet etablerat, varefter allt är klart för dataöverföring.

Tillämpningssambandet kan avvecklas på tre olika sätt. Normal avvecklig sker med

hjälp av tjänsten A-RELEASE. Det är också en bekräftad tjänst. Ur respondentens svar framgår huruvida den godtar en avveckling av sambandet eller inte. Denna bekräftade avveckling kallas också för *ordnad avveckling* (orderly release).

En mera abrupt tjänst för att avveckla sambandet är det av *användaren initierade avbrottet* (user-initiated abort) A-ABORT. A-ABORT är en icke-bekräftad tjänst; initiatorn meddelar att sambandet är brutet och avslutar dialogen. Respondenten underrättas därom i form av primitivet A-ABORT.indication.

Om någon funktion i de undre skikten inte kan uppfyllas kan detta ge upphov till ett av *tillhandahållaren initierat avbrott* (provider-initiated abort). Den tjänsten betecknas A-P-ABORT och består av endast ett primitiv, A-P-ABORT.indication, som ges åt sambandets bägge tillämpningsenheter.

ACSE:s protokoll omfattar fem protokolldataenheter. Tjänsteprimitivet A-ASSOCIATE.request resulterar i att protokolldataenheten AARQ sänds till mottagarsystemet, där den ger upphov till indikationen A-ASSOCIATE.indication. Respondentens svar (A-ASSOCIATE.response), genererar i sin tur protokolldataenheten AARE som, då den nått partnerenheten, har till följd att initiatorn levereras tjästeprimitivet A-ASSOCIATE.indication.

Även tjänsten A-RELEASE begagnar sig av två protokolldataenheter för att uföra sitt värv: RLRQ och RLRE, medan den obekräftade tjänsten A-ABORT ger upphov till endast en protokolldataenhet: ABRT.

ACSE:s tjänsteprimitiv och protokolldataenheter är sammanfattade i tabell 1.

Tjänst	Primitiv	APDU
A-ASSOCIATE	A-ASSOCIATE.request	AARQ
	A-ASSOCIATE.indication	
	A-ASSOCIATE.response	AARE
	A-ASSOCIATE.confirmation	
A-RELEASE	A-RELEASE.request	RLRQ
	A-RELEASE.indication	
	A-RELEASE.response	RLRE
	A-RELEASE.confirmation	
A-ABORT	A-ABORT.request	ABRT
	A-ABORT.indication	
A-P-ABORT	A-P-ABORT.indication	—

Tabell 1. ACSE tjänsteprimitiv och protokolldataenheter

2.5.2 Tillämpningselementet för fjärroperationer — ROSE

En fjärroperation är en form av proceduranrop från ett program i ett system till ett program i ett annat system. För att möjliggöra fjärroperationer mellan öppna system måste förutom själva datakommunikationen också sättet på vilket operationerna är specificerade standardiseras. I OSI modellen ingår en sk. *RO-notation* [Rose 1990 s. 420] för fjärroperationer. Den är definierad med hjälp av makrospråket i ASN.1. Med RO-notationen specificerar man gränssnittet till en fjärroperation, vilka argument som ingår och hurudana resultat- och felmeddelanden den kan ge upphov till. Fjärroperationer är ett sätt på vilket distribuerad databehandling kan realiseras.

En begäran är det enklaste fallet av en fjärroperation. I vissa fall ger den upphov till ett svar från fjärrsystemet. En begäran av en fjärroperation består av åtminstone fyra parametrar som vidarebefordras till fjärrsystemet:

- Ett *operationsnummer* (operation value). Det indikerar vilken operation som skall utföras.
- Ett *argument* (argument) av godtycklig komplexitet. Ifall operationen inte har några argument kan parametern fattas.
- En *anropsidentifikation* (invocation identifier). Ett löpande nummer som används för att identifiera anropen.
- En *länkad anropsidentifikation* (lindek invocation identifier) som används om operationen utgör en del av en större fjärroperation (jfr. underprogram).

Dessutom finns det två lokala parametrar, *operationsklass* (operation class) och *prioritet*, (priority) för att styra de underliggande skikten. De överförs inte till fjärrsystemet.

Om operationen kan utföras och ger upphov till ett resultat sänder det anropade systemet resultatet till anroparen. Resultatet innehåller anropsidentifikationen och möjligen en dataparameter. Ifall operationen inte kan utföras returnerar det anropade systemet ett felmeddelande som består av anropsidentifikationen, en felkod och en dataparameter.

I vissa fall, t.ex. om det anropade systemet inte känner till operationen, kan fjärroperationen förkastas eller underkännas. Anroparen får ett meddelande som innehåller anropsidentifikationen och varför den underkänts.

För varje operation specificeras typen av argument och hurudana resultat- och felmeddelanden som kan väntas. Detta sker med hjälp av RO-notationen. Hur en operation internt utförs beskrivs inte.

Tillämpningselementet ROSE (Remote Operations Service Element) [ISO 1989b, 1989c] är ett symmetriskt tillämpningselement som tillhandahåller tjänster för att såväl initiera fjärroperationer som för att ta emot anrop från fjärrsystem. De tjänsteprimitiv som ROSE tillhandahåller börjar alla med prefixet RO-. Det är skäl att notera, att man via ACSE först måste etablera ett samband innan man kan utnyttja de tjänster ROSE erbjuder.

För att anropa en fjärroperation använder man sig av den icke-bekräftade tjänsten RO-INVOKE. Om operationen kan utföras sänder fjärrsystemet resultatet till initiatorn med hjälp av tjänsten RO-RESULT. Ifall operationen av någon orsak misslyckas anropar fjärrsystemet tjänsten RO-ERROR. Såväl RO-RESULT som RO-ERROR är icke-bekräftade tjänster.

Om en operation förkastas meddelas initiatorn därom med hjälp av den icke-bekräftade tjänsten RO-REJECT-U. Tjänsten RO-REJECT-P används om något underliggade skikt försummar att tillhandahålla de avtalade tjänsterna. Bägge fjärrsystemen får då primitivet RO-REJECT-P.indication.

ROSE:s tjänsteprimitiv och protokolldataenheter är sammanfattade i tabell 2.

Tjänst	Primitiv	APDU
RO-INVOKE	RO-INVOKE.request	ROIN
	RO-INVOKE.indication	
RO-RESULT	RO-RESULT.request	RORS
	RO-RESULT.indication	
RO-ERROR	RO-ERROR.request	ROER
	RO-ERROR.indication	
RO-REJECT-U	RO-REJECT-U.request	RORJ
	RO-REJECT-U.indication	
RO-REJECT-P	RO-REJECT-P.indication	—

Tabell 2. ROSE tjänsteprimitiv och protokolldataenheter

Nedanstående exempel visar hur en fjärroperation `dividera` för heltalsdivision av två heltal kunde definieras med hjälp av RO-notation. Fjärroperationens operationsnummer är 7. Argumentet är av typen `Talpar` som består av två heltal, `taljare` och `namnare`, vilka båda är `INTEGER`. Resultatet är också ett heltal `INTEGER`. Ifall nämnaren är noll returneras felmeddelandet `namnareNoll`, som har värdet 2.

```
dividera OPERATION
    ARGUMENT      Talpar
    RESULT        INTEGER
    ERRORS        { namnareNoll }
::= 7

Talpar ::= SEQUENCE {
                  namnare INTEGER,
                  taljare INTEGER
           }

namnareNoll       ERROR ::= 2
```

Mallarna för tjänsteprimitiven RO-INVOKE.request och RO-RESULT.request ser ut så här:

```
RO-INVOKE.req (operVal, operClass, arg, invID, lnkID, prior)
RO-RESULT.req (invID, result, operVal, prior)
```

Ifall två tal, t.ex. 21 och 5, skall divideras med varandra i ett fjärrsystem fås

nedanstående anrop. De optionella parametrarna har ersatts av tecknen för kommentar i ASN.1.

```
tva_tal Talpar ::= { 21, 5 }
RO-INVOKE.req (7, ----, tva_tal, 3, ----, ----)

svar INTEGER ::= { 4 }
RO-RESULT.req (3, svar, 7, ----)
```

Parametrarna `tva_tal` och `svar` ingår egentligen inte som sådana i RO_INVOKE respektive RO_RESULT. De är ur ROSE:s synvinkel transparent användardata som skall överföras från system till system. Deras semantiska tolkning överlåts åt tillämpningsprocessen.

3. OSI driftstöd

Ett mindre självständigt nätverk, t.ex. ett lokalnät i ett kontor, fungerar i allmänhet relativt väl utan desto vidlyftigare underhåll. Datamaskiner, skivminnen, skrivare o.dyl. ansluts alla som självständiga element till nätet. Om ett nätelement mot förmodan är ur funktion märker man det ofta först när man behöver det. Då brukar det räcka att man går till det och startar det på nytt, varefter nätet igen erbjuder de tjänster man behöver.

Stora och komplicerade nätverk, som t.ex. ett regionomfattande telefonnät eller ett radionät, kan nauturligtvis inte styras och administreras enligt en sådan "låt gå" mentalitet. Nätets geografiska utbredningen gör att man inte kan ha alla nätelement inom räckhåll för lokal felsökning. Betalande användare tolererar inte stora kast i den erbjudna servicenivån. De stora investeringarna och rörliga kostnaderna som är förknippade med ett sådant nät förutsätter att driften är optimal.

Termen *driftstöd* (network management) är en sammanfattande benämning på de aktiviteter som är ägnade att bl.a:

- hålla nätets verkningsgrad hög
- hålla nätets servicenivå hög
- hålla nätets driftskostnader låga
- upprätthålla säkerheten i nätet.

Eftersom data- och telekommunikationsnäten hela tiden växer i omfång och komplexitet är det klart att behovet av driftstöd blir allt större. För att ett driftstödsystem skall vara effektivt måste det vara flexibelt, leverantörsoberoende och tillräckligt generellt för att kunna administrera och övervaka diversifierad apparatur. Dessutom måste även abstrakta helheter som förbindelser och kostnadsuppföljning kunna innefattas i driftstödet.

Internationella gremier som ISO och IAB (Internet Activities Board) har redan länge sysslat med standardisering av driftstöd. Denna hänför sig inte enbart till administration av kommunikationsnät. I synnerhet ISO:s koncept för driftstöd baserar sig på en generell referensmodell för övervakning och administration. I detta diplomarbete kommer tyngdpunkten av naturliga skäl dock att ligga på övervakning av kommunikationsnät och -tjänster. Standarden tar i första hand inte ställning till *vad* som skall övervakas, utan tillhandahåller metoder för *hur* sk. styrda objekt kan beskrivas. Med hjälp av ett driftstödsnät, som i allmänhet bara är ett logiskt nät, kan dessa objekt sedan manipuleras.

ISO har standardiserat såväl ett tillämpningselement som ett protokoll för driftstöd kallade CMISE (Common Management Information Service Element) respektive CMIP

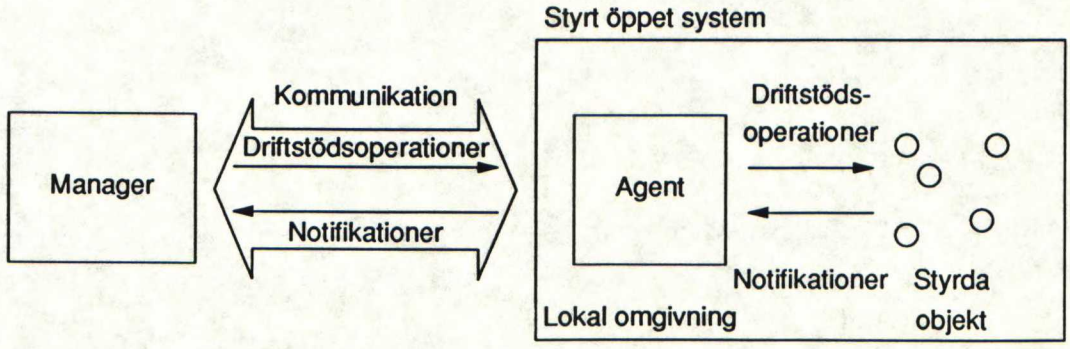
(Common Management Information Protocol) [ISO 1990a, 1990b].

3.1 Driftstöd — en allmän överblick

Den del av ett eller flera nätverk som administrativt sett är en helhet kalls en *driftstödsdomän* (network management domain). I den kan det finnas ett godtyckligt antal *nätelement* (network element), som t.ex. modem, bryggor och korskopplare. Om ett nätelement kan övervakas och styras av en driftstödstillämpning kallas den för ett *styrt objekt* (managed object).

Den övervakande enheten i nätet kallas för *manager*. Managern kommunicerar i allmänhet inte direkt med de styrda objekten utan gör det via en sk. *agent*. Managern och agenten kommunicerar med hjälp av protokollet CMIP, medan agentens och de styrda objektens samverkan inte specificerats i standarden. Om de styrda objekten är modem kan t.ex. AT-kommandon [NDC 1990] användas. Ingenting hindrar att det i en driftstödsdomän finns flera managers som oberoende av varandra kommunicerar med flera agenter.

Managern beordrar agenten att utföra en *driftstödsoperation* (management operation) på önskade styrda objekt. Oberoende av denna kommunikation kan ett styrt objekt sända managern en *notifikation* (notification) via agenten. Förhållandet mellan dessa begrepp framgår ur figur 8.



Figur 8. Manager, agent och styrda objekt [OSI/NM Forum 1989]

Driftstöd enligt ISO är ingalunda begränsat till övervakning och administration av kommunikationsnät och -komponenter. Det går alldeles väl att använda CMIP till att styra vad som helst t.ex. nättjänster och -tillämpningar. En förutsättning är förstås att de styrda objekten är definierade i enlighet med standarderna för driftstöd.

Om en manager vill utföra en driftstödsoperation på ett bestämt styrt objekt måste den specificera följande fyra saker:

- det styrda objektets *klass*
- det styrda objektets *namn*
- *operationen* som skall utföras
- *parametrarna* som hör till operationen.

Ett exempel på en driftstödsoperation där man beordrar ett modem att öppna en förbindelse med telefonnumret 123 456 kunde vara:

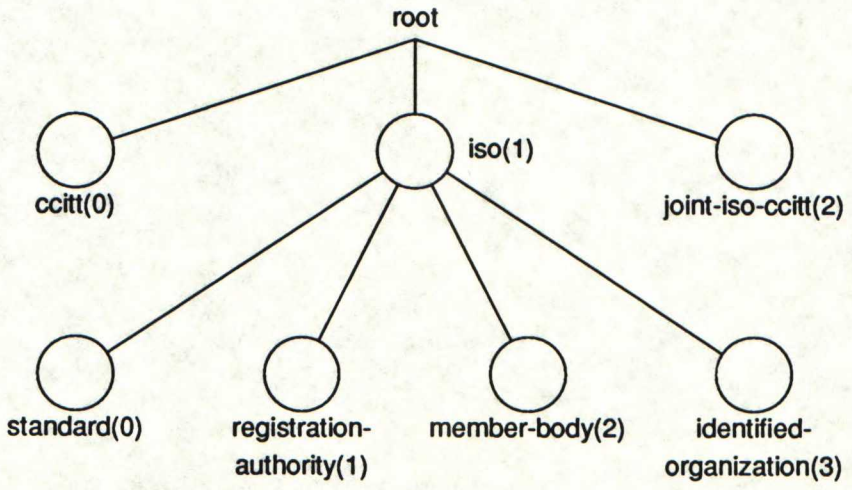
objektklass:	klassen modem
objektnamn:	modem nummer 3
operation:	ring
parameter:	123 456

För att flexibelt kunna beskriva ett objekt begagnar man sig av objekt-orienterade definitionsmetoder. Ett objekt är en sluten enhet vars interna struktur är väl förborgad; endast de egenskaper som definierats synliga är åtkommliga utifrån. Via dessa kan man samverka med objektet. Varje objekt måste tillhöra någon klass.

Objektklasser kan definieras genom att man förfinar andra, redan existerande objektklasser. I denna arvshierarki ärver en objektklass alla egenskaper från en överordnad objektklass och ges dessutom några egenskaper till. Den översta objektklassen, från vilken alla andra objektklasser åtminstone indirekt är härledda, kallas TOP.

En objektklass för driftstöd består av dess utåt synliga attribut, de operationer man kan göra på objekten (förekomsterna av denna objektklass) och hur objekten härvidlag reagerar samt de notifikationer objekten kan sända. Alla objekt som tillhör samma klass har samma attribut, operationer, beteende och notifikationer.

Man strävar till att standardisera objektklasserna på global nivå. I det sk. *registreringsträdet* (se figur 9) registreras allehanda informationsobjekt, bl.a. objektklasser för driftstöd. Alla registrerade informationsobjekt har en unik ASN.1-objektidentifierare.

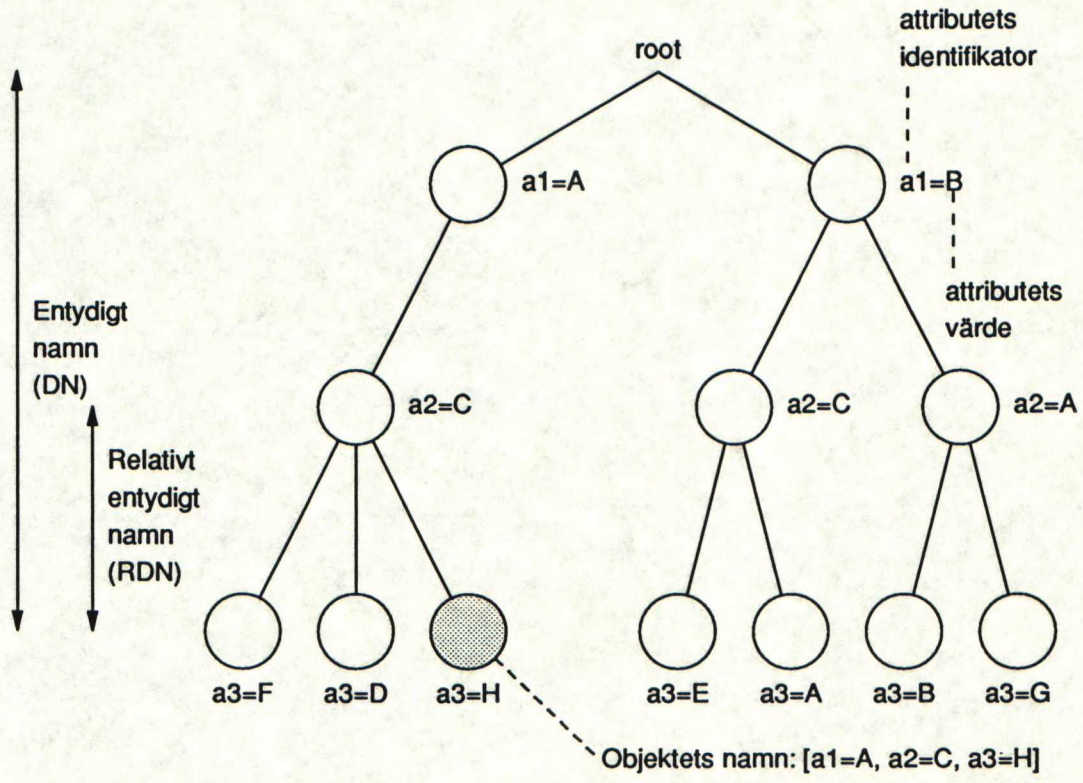


Figur 9. Registreringsträdet för objektidentifierare [Kärkkäinen 1991]

En nod i trädet, t.ex. ISO, kan delegera registreringsrättigheter nedåt i sitt underträd. Under noden member-body finns bl.a. Finlands Standardiseringsförbund SFS som i sin tur har rätt att distribuera sin registreringsauktoritet i sitt underträd.

De styrda objektens namn registreras i ett för driftstödsdomänen lokalt namnträd (Management Information Tree, MIT). Precis som i registreringsträdet kallas även här det översta objektet rot (root) och innehåller alla underordnade objekt. Ett objekt har ett entydigt relativt namn (Relative Distinguished Name, RDN) i sitt överordnade objekt, som i sin tur har ett RDN i sitt överordnade objekt osv. Ett objekts entydiga namn (Distinguished Name, DN) består av det överordnade objektets entydiga namn och sitt eget relativa entydiga namn.

I figur 10 är ett namnträd för styrda objekt i en driftstödsdomän avbildat. Det skuggade objektets relativa entydiga namn (RDN) är `[a3=H]`. Dess entydiga namn (DN) är `[a1=A, a2=C, a3=H]`.



Figur 10. De styrda objektens namnträd [Kärkkäinen 1991]

Genom att i en driftstödsoperation specificera dess *utsträckning* (scope) eller *filtrering* (filtering) kan man välja vilka objekt man opererar på. Utgående från ett basobjekt kan man specificera till vilka underordnade objekt operationen utsträcks (t.ex. basobjektet och hela dess underträd). Filtrering möjliggör ett urval av objekt på basen av vissa villkor; i figur 10 finns det t.ex. två objekt som uppfyller villkoret `[a2=C]`.

3.2 Beskrivning av styrda objekt

Objektklasserna som registreras i trädet för objektidentifierare måste naturligtvis specificeras med ett formellt beskrivningsspråk. OSI/NM Forum, en sammanslutning av leverantörer inom data- och telebranschen, har ett eget underträd i registreringsträdet. Dess objektidentifierare är `{iso member-body(2) canada(124) osi/nm-forum(360501)}`. Under denna nod i trädet registrerar OSI/NM Forum sina egna objektklasser för driftstöd.

OSI/NM Forum har gjort ett eget beskrivningsspråk med hjälp av ASN.1 makrodefinitioner som kallas Forum Object Specification Framework. Utan att desto vidare gå in på dess formella syntax presenteras här nedan ett exempel på hur en objektklass för ett enkelt modem kunde se ut i denna notation.

```
modem M-OBJECT-CLASS
    DERIVED FROM                {top}
    BEHAVIOUR DEFINITIONS       -- see clause 1.41/2.72/3.14
    CHARACTERIZED BY
        ATTRIBUTES
            MUST CONTAIN        {
                                administrativeState,
                                modemSystemID READ-ONLY
                                }
            MAY CONTAIN         {
                                autoAnswerEnable,
                                manufacturerID READ-ONLY
                                }
        OPERATIONS
            CREATE
            DELETE
            ACTIONS             {
                                dialConnect,
                                dialDisconnect,
                                selfTest
                                }
            NOTIFICATIONS       {
                                busy,
                                connected,
                                noCarrier,
                                noDialtone
                                }
::= {modem-objectClass 1}
```

I beskrivningen av objektklassen definieras dess utåt synliga attribut, operationer och notifikationer.

Operationerna `CREATE` och `DELETE` betyder ingalunda att man via lämpliga driftstödsoperationer kan tillverka eller förstöra ett modem. En beskrivning av en objektklass i vilken dessa operationer inte ingår är ägnad att enbart fungera som en interim klass som skall vidareförfinas. Sålunda kunde en klass `telekommunikationsutrustning` vara en klass vars enda syfte är att man ur den härleder klasserna `modem` och `telefoncentral`.

Ovanstående definition är naturligtvis inte tillräcklig, utan attributen, operationerna och notifikationerna måste i sin tur beskrivas i detalj. I följande kodfragment har några komponenter ur objektklassen `{modem-objectClass 1}` brutits ner i sina beståndsdelar:

```
administrativeState ATTRIBUTE
    WITH ATTRIBUTE SYNTAX AdministrativeState
    ::= {modem-attribute 1}

modemSystemID ATTRIBUTE
    WITH ATTRIBUTE SYNTAX INTEGER
    ::= {modem-attribute 2}

dialConnect ACTION
    ACTIONINFO PhoneNumber
    ::= {modem-action 1}

connected NOTIFICATION
    EVENTINFO BitRate
    ::= {modem-notification 3}

AdministrativeState ::= ENUMERATED {
                            locked          (0),
                            unlocked        (1),
                            shuttingDown    (2)
                            }

PhoneNumber ::= PrintableString (SIZE(0..31))

BitRate ::= ENUMERATED {
                bps1200     (0),
                bps2400     (1),
                bps4800     (2),
                bps9600     (3)
                }
```

För att kunna specificera vilket styrt objekt man vill manipulera måste man hierarkiskt binda namn till objekten i ett namnträd (MIT). Genom att i en driftstödsoperation specificera objektets entydiga namn (DN) eller dess relativa entydiga namn (RDN) i förhållande till något annat objekt utförs den önskade handlingen på det tänkta objektet. Sålunda kunde vårt modem från tidigare exempel kallas `(N=Nät-1, A=Avdelning-2, modemSystemID=3)`.

Då driftstödstillämpningen (managern) anser att den bör utföra en operation på vårt

exempelmodem etablerar den en förbindelse med agenten (via ACSE), sänder det önskade driftstödskommandot (via CMISE) och bryter sedan förbindelsen (via ACSE). I driftstödskommandot specificerar managern objektklass, namn, operation och eventuella paramerar.

Agenten tar emot driftstödskommandot och lokaliserar den utrustning som avses. Därefter omvandlar agenten kommandot i en utrustnings- eller leverantörspecifik form (t.ex. AT-kommandon [NDC 1990] för modem) och utför operationen. Själva arbetet att styra apparaturen utförs alltså i agenten.

Om vi återupplivar vårt ursprungliga exempel (att öppna en modemförbindelse till telefonnummret 123 456) får det nu följande form:

objektklass:	`{modem-objctClass 1}`
objektnamn:	`(N=Nät-1, A=Avdelning-2, modemSystemID=3)`
operation:	`{modem-action 1} -- dialConnect`
parameter:	`123456`

På basen av objektklassen och -namnet lokaliserar agenten med hjälp av någon lokal namnserver det önskade modemet. Operationens (`dialConnect` med parametern `123456`) motsvararighet i AT-kommandomängden är strängen `ATD123456`. Den sänder agenten till modemet.

Då modemet i andra ändan svarar och förbindelsen öppnas sänder det lokala modemet strängen `CONNECT 9600` som indikation därom till agenten. Agenten sänder i sin tur notifikationen `connected ({modem-notification 3})` med parametern `bps9600` till managern.

3.3 Tillämpningselementet för driftstöd — CMISE

CMISE (Common Management Information Service Element) [ISO 1990a] är ett allmänt tillämpningselement som möjliggör överföring av driftstödsinformation mellan kommunicerande enheter. CMISE använder sig av ACSE:s [ISO 1988a] tjänster för att etablera och bryta samband och av ROSE:s [ISO 1989b] tjänster för att låta utföra driftstödsoperationerna.

Protokollet som reglerar utbytet av dataenheter mellan de kommunicerande enheterna kallas CMIP (Common Management Information Protocol) [ISO 1990b].

Etablering av samband, som alltid måste ske innan driftstödsoperationer kan utföras, sker genom tjänsten M-INITIALIZE. För avveckling och brytning av förbindelse

används M-TERMINATE respektive M-ABORT. M-ABORT är tillsammans med M-EVENT-REPORT de enda icke-bekräftade tjänsterna som CMISE erbjuder.

För att skapa och avlägsna förekomster av dataobjekt som representerar verkliga styrda objekt används tjänsterna M-CREATE och M-DELETE.

Tjänst	Primitiv	APDU
M-INITIALIZE	M-INITIALIZE.req/ind	INIRQ
	M-INITIALIZE.resp/conf	INIRE
M-ACTION	M-ACTION.req/ind	ACTINV
	M-ACTION.resp/conf	ACTRES
M-CREATE	M-CREATE.req/ind	CREINV
	M-CREATE.resp/conf	CRERES
M-DELETE	M-DELETE.req/ind	DELINV
	M-DELETE.resp/conf	DELRES
M-EVENT-REPORT	M-EVENT-REPORT.req/ind	ERINV
	M-EVENT-REPORT.resp/conf	ERRES
M-GET	M-GET.req/ind	GETINV
	M-GET.resp/conf	GETRES
M-CANCEL-GET	M-CANCEL-GET.req/ind	CAGINV
	M-CANCEL-GET.resp/conf	CAGRES
–	–	LRINV
M-SET	M-SET.req/ind	SETINV
	M-SET.resp/conf	SETRES
M-TERMINATE	M-TERMINATE.req/ind	TERRQ
	M-TERMINATE.resp/conf	TERRE
M-ABORT	M-ABORT.req/ind	ABRT

Tabell 3. CMISE tjänsteprimitiv och protokolldataenheter

M-ACTION är tjänsten som används för att låta utföra driftstödsoperationer, medan man med tjänsterna M-SET och M-GET kan läsa och manipulera objektens attribut.

Tjänsterna M-ACTION, M-DELETE, M-GET och M-SET kan förses med sådana utsträcknings- eller filtreringsparametrar att de verkar på flera objekt samtidigt. Om man t.ex. utför tjänsten M-GET på en grupp objekt överförs svaren för varje objekt skilt för sig i en länkad PDU kallad LRINV. Då alla svar överförts sänder agenten PDU:n

GETRES med tom dataparameter som tecken på att operationen är avslutad. Ifall man väljer för stor utsträckning för tjänsten M-GET kan man avbryta den med tjänsten M-CANCEL-GET.

Notifikationer, slutligen, överförs med hjälp av tjänsten M-EVENT-REPORT.

CMISE:s tjänsteprimitiv och protokolldataenheter är sammanfattade i tabell 3.

CMIP är ett allmänt protokoll för överförande av driftstödsinformation. Det är ingalunda begränsat till tele- eller datakommunikationstillämpningar. Förutsatt att nödvändiga objektklasser är definierade kunde man mycket väl tänka sig att följande operation inom gebitet för statsförvaltning:

objektklass:	självständig nation
objektnamn:	Finland
operation:	ansök om medlemskap i EG
parameter:	lämpliga förbehåll

Innan vi är mogna för en statsförfattning enligt CMIP bör vi nog utveckla en standard för fördelning av politiskt ansvar.

4. Protokollutvecklingsverktyget CVOPS

Stora kommunikationssystem är komplexa helheter. Att implementera en OSI-konform protokollstapel med hjälp av endast redigerare och kompilator är ingen avundsvärd uppgift. För att avhjälpa härmed förknippade svårigheter har olika halvautomatiska hjälpmedel utvecklats.

Datakommunikationsprotokoll kan i allmänhet beskrivas med något formellt språk. Det är en avsevärd fördel om man i protokollutvecklingsomgivningen kan använda sig av t.ex. tillståndstabeller som ofta förekommer i ISO-standarder. I gynsamma fall kan man dessutom begagna sig av automatisk generering av programkod.

Förutom att olika typer av hjälpmedel underlättar själva utvecklingsarbetet kan de också möjliggöra automatisk verifiering av implementationen.

År 1983 påbörjades arbetet på protokollutvecklingsverktyget VOPS (Virtual OPerating System) vid Statens tekniska forskningscentral. VOPS utvecklades ursprungligen i Pascal. Senare skrevs det om i programmeringsspråket C, och fick i samband med det namnet CVOPS (C language Virtual OPerating System) [CVOPS 1990].

Med hjälp av CVOPS kan man specificera, simulera, implementera och testa datakommunikationsprotokoll skrivna i programmeringsspråket C i en arbetsstation.

4.1 Virtuellt operativsystem

CVOPS är ett virtuellt operativsystem som består av *virtuella uppdrag* eller *virtualuppdrag* (virtual task eller vtask). Virtualuppdragen, som motsvarar skiktenheter eller tillämpningselement i OSI-stacken, kommunicerar med varandra genom att utbyta meddelanden. Deras logik beskrivs i så kallade *utvecklade, ändliga tillståndsautomater* (EFSA, Extended Final State Automaton).

I initialiseringsfasen läser CVOPS virtualuppdragens automater och övriga specifikationsfiler samt översätter protokollspecifikationerna till ett internt format. Man behöver med andra ord inte beskriva beteendet i C, utan kan använda sig av ett för ändamålet enkom utvecklat specifikationsspråk. Dessutom ingår ett antal i C skrivna stödfunktioner i varje virtualuppdrag. CVOPS' kärna tillhandahåller allmänna tjänster som virtualuppdragen utnyttjar. I kärnan finns det också programkod för administrering av de olika virtualuppdragen.

Programkoden för virtualuppdragen och CVOPS kompileras och länkas till ett exekverbart program. Sett ur värdoperativsystemets synvinkel är kommunikationsprogrammet en applikationsprocess bland andra. Internt är det dock CVOPS' kärna som tar

kommandot och låter virtualuppdragen utföra sina protokoll.

Till kärnans uppgifter hör att i initaliseringsskedet bygga upp en intern representation av protokollstacken och sedan fördela körturer åt virtualuppdragen. Ett virtualuppdrag kommunicerar med ett annat genom att asynkront sända ett internt meddelande. Beroende på meddelandets prioritet läggs det i en av tre FIFO-köer. CVOPS väljer det första (äldsta) meddelandet av möjligast hög prioritet och anropar det mottagande virtualuppdragets basfunktion. Virtualuppdraget utför nu de handlingar som finns beskrivna i automaten. Den kan bl.a. sända meddelanden till andra virtualuppdrag, byta tillstånd eller förändra värdet på en intern variabel. Då virtualuppdraget utfört sin uppgift återvänder kontrollen till CVOPS, som väljer nästa meddelande från en FIFO-kö och igen anropar en basfunktion. Om alla FIFO-köer är tomma väntar CVOPS på ett externt meddelande från värdoperativsystemet eller ett timeout-meddelande från klockmodulen.

För att CVOPS skall kunna kommunicera med värdoperativsystemet behövs en operativsystemspecifik *inmatningsmodul* och speciella *drivuppdrag* för gränssnitt. I inmatningsmodulen, som länkas till CVOPS' kärna, finns rutiner som stöder kommunikationen mellan värden och CVOPS. De ovan nämnda externa meddelandena kommer till CVOPS via ett drivuppdrag.

Ett drivuppdrag ser till det yttre ut som ett vanligt virtualuppdrag, men det har en speciell funktion. Det konverter interna meddelanden till en på förhand specificerad form och sänder dem t.ex. till en kommunikationsport. Drivuppdraget utnyttjar direkt värdoperativsystemets tjänster för interprocesskommunikation.

Drivuppdrag används i allmänhet vid portering av en protokollstack genom att kapsla in protokollstacken mellan två operativsystemspecifika drivuppdrag.

CVOPS tillhandahåller inmatningsmoduler och drivuppdrag till vissa standardiserade gränssnitt såsom sockets och rör i UNIX. Inom ramen för detta diplomarbete gjorde författaren en inmatningsmodul för DNOS (*dnosinput*) och drivuppdrag till protokollstackens bägge ändor (*hdlcif* och *cmif*).

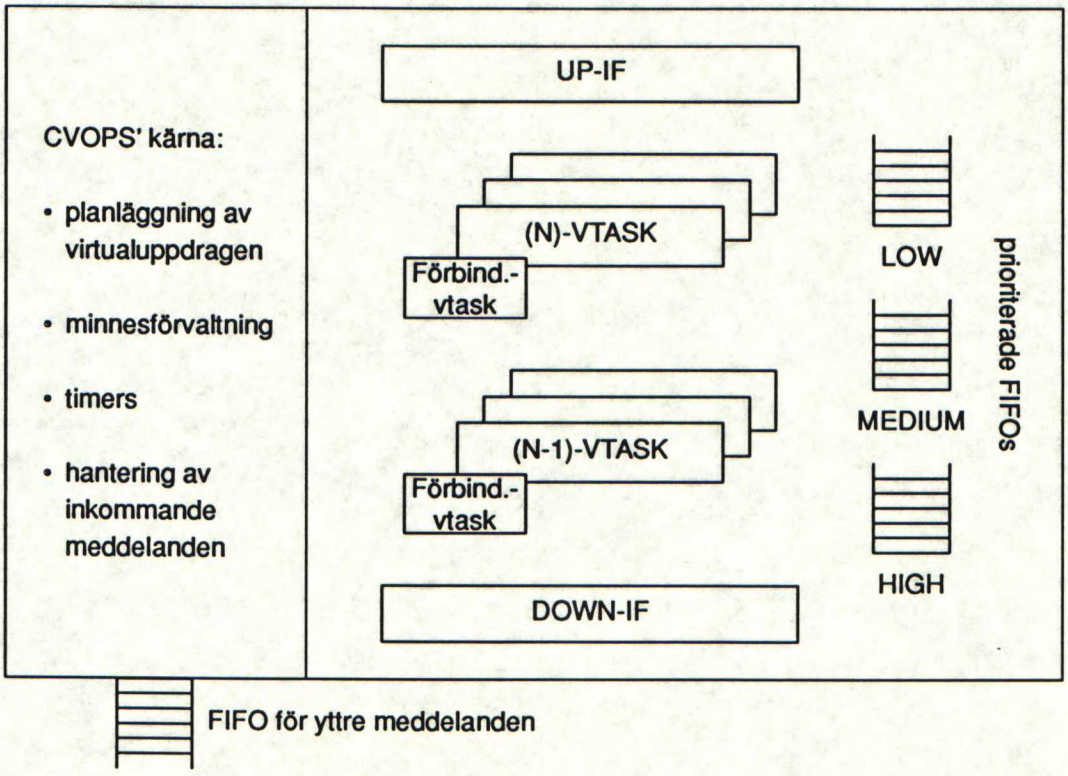
Precis som OSI-modellen stöder även CVOPS flerkanalprotokoll, dvs. flera parallella, oberoende förbindelser i samma skikt. Varje förbindelse ombesörjs av ett eget virtualuppdrag, ett så kallat *förbindelsevirtualuppdrag* (connection vtask). Den representerar förbindelseändpunkten (CEP). Dessutom finns det ett för hela skiktet gemensamt *enhetsvirtualuppdrag* (entity vtask) som fungerar som "trafikpolis". Alla meddelanden som kommer till detta skikt ges åt enhetsvirtualuppdraget, som på basen av meddelandets förbindelseändpunktsidentifikation vidarebefodrar meddelandet till rätt förbindelsevirtualuppdrag.

Då en förbindelse etableras väljer enhetsvirtualuppdraget en ledig förbindelseuppgift till vilken det sänder meddelandet.

Figur 11 visar en enkel, flerkanalig protokollstack bestående av två skikt, som i god

OSI-anda kallas (N) och (N+1). Dessa skikt realiseras i form av virtualuppdrag. För varje förbindelse finns det ett förbindelsevirtualuppdrag. I varje skikt finns det ett enhetsvirtualuppdrag.

Gränssnittet från CVOPS till operativsystemet finns i de två drivuppdragen, medan inmatningsmodulen befinner sig i kärnan. I kärnan finns också rutiner för minnesförvaltning och klocktjänster.



Figur 11. CVOPS och virtuella uppdrag [Ahtiainen 1990 s. 345]

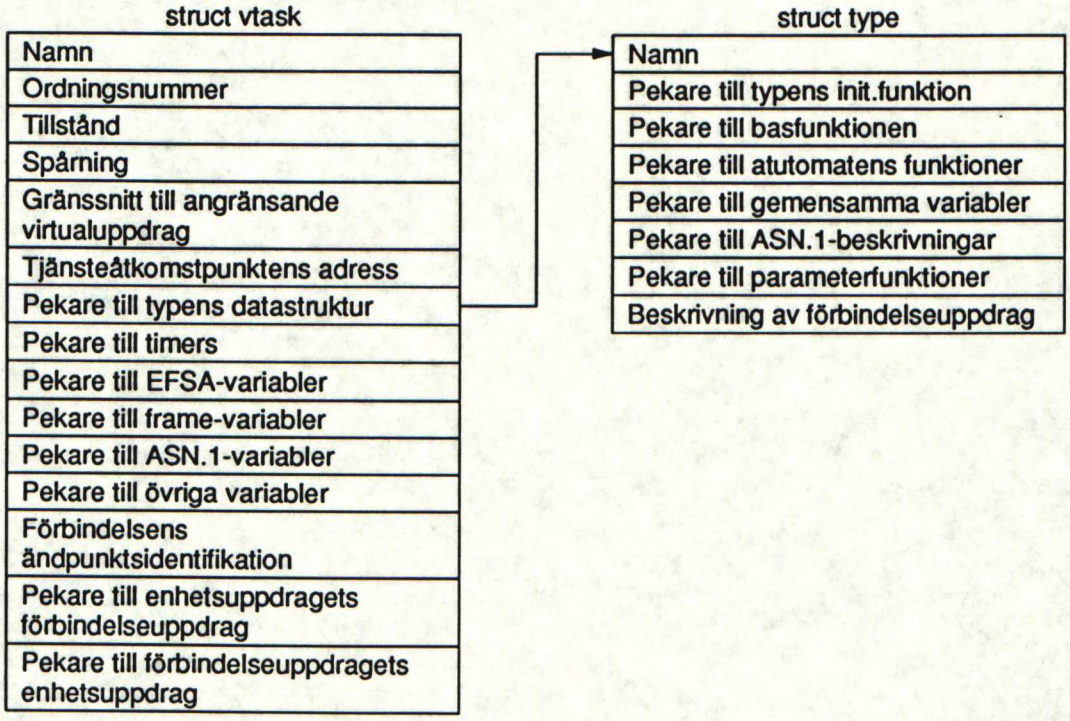
Eftersom CVOPS är en enda process i värdoperativsystemet har alla virtualuppdrag en gemensam minnesrymd. Vid kommunikation mellan virtualuppdragen räcker det alltså med att överföra en pekare till meddelandet. På så vis undviker man onödig kopiering av minnesblock.

4.2 Virtuella uppdrag

De från OSI-modellen bekanta protokollenheterna och applikationstjänsteelementen realiseras i CVOPS i form av virtuella uppdrag. Gemensamt för alla virtuella uppdrag är

den yttre struktur CVOPS ger dem. Internt är de alla naturligtvis olika — de har ju olika funktion att fylla — men CVOPS förutsätter att ett virtualuppdrag har en viss uppbyggnad.

CVOPS är en utvecklingsmiljö i programmeringsspråket C. De centrala datastrukturerna som hänför sig till ett virtualuppdrag är av arten `struct`. Nedanstående figur 12 beskriver strukturerna för ett virtualuppdrag (`struct vtask`) och dess typ (`struct type`).



Figur 12. Virtualuppdraget och dess typ

Ett virtualuppdrags beteende finns beskrivet i dess typ. Protokollets logik, parameterfunktionerna och kodningsreglerna definieras alla i typen. Förbindelsevirtualuppdragen i ett skikt har en gemensam `struct type` men egna `struct vtask`.

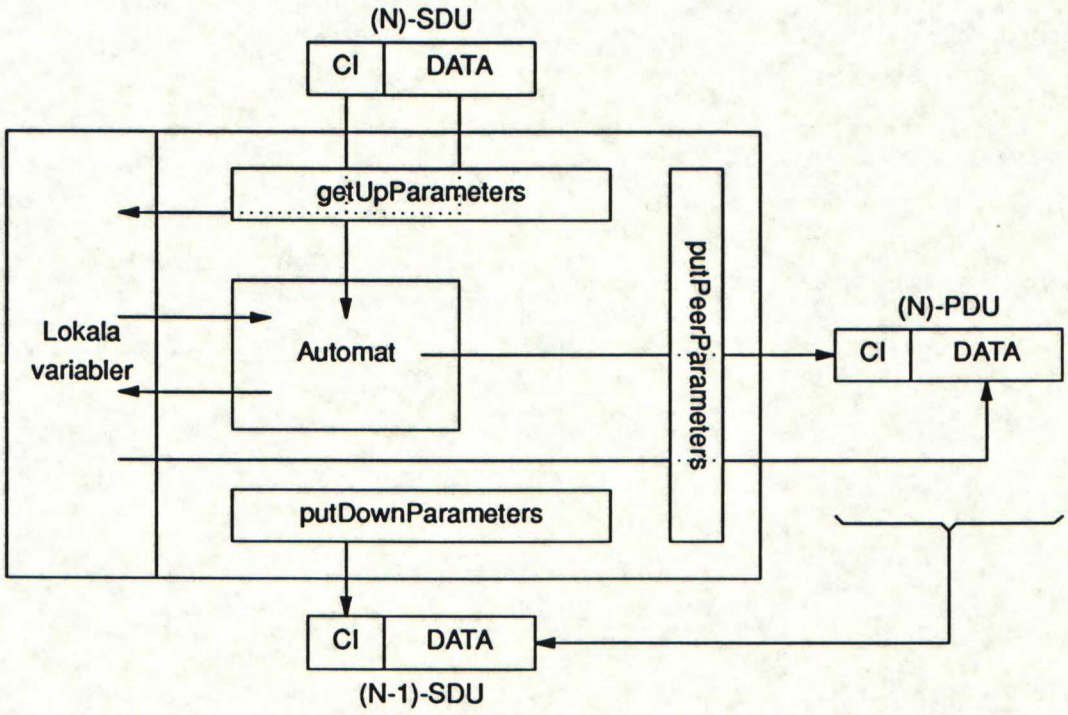
I normala fall har ett virtualuppdrag (minst) tre gränssnitt: UP mot det övre skiktets virtualuppdrag, DOWN mot det nedre skiktets virtualuppdrag och PEER mot partnerenheten. I virtualuppdraget har varje gränssnitt två speciella funktioner: `get*Parameters` och `put*Parameters`. Den förstnämnda undersöker ett inkommande meddelande och extraherar parametervärden ur det, medan den sistnämnda skriver parametervärden i ett utgående meddelande.

Figur 13 beskriver de olika parameterfunktionernas roll i ett virtualuppdrag. Pilarna visar dataflödet både i och utanför virtualuppdraget.

Ett tjänsteprimitiv kommer till virtualuppdraget via UP-gränssnittet i form av ett meddelande. Funktionen `getUpParameters` läser parametrarna i den inkommna (N)-SDU:n och lagrar dess värden i interna variabler. Tjänsteprimitivets styrinformation (CI) ges till automaten, som utför protokollets funktion.

Om automaten skall sända en PDU till sin partnerenhet anropar den funktionen `putPeerParameters`, på vars ansvar kodningen av parametrarna ligger.

För att sända den nyss kodade PDU:n paketeras den tillsammans med övriga (N-1) tjänsteparametrar i en (N-1)-SDU och ges till det underliggande virtualuppdraget som ett tjänsteprimitiv. Den här manövern ombesörjs av funktionen `putDownParameters`, som förutom parametrarna också ger (N-1)-SDU:n dess CI.

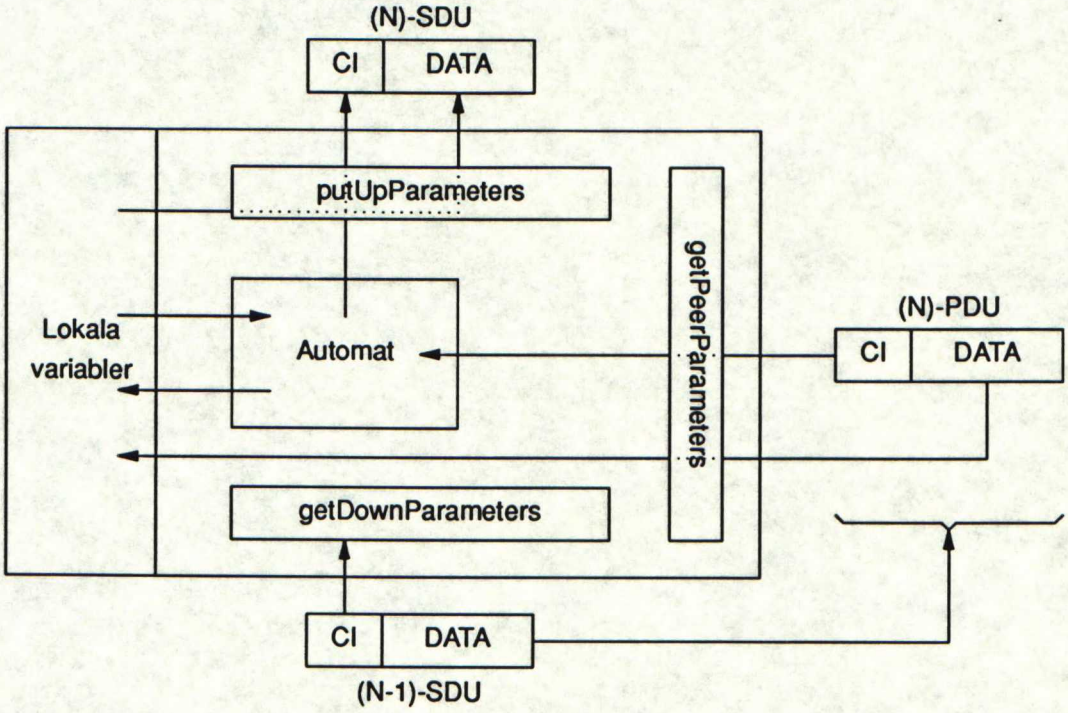


Figur 13. Meddelande till ett virtualuppdrag via UP-gränssnittet [Ahtiainen 1990 s. 346]

På detta sätt åker meddelandet nedåt i protokollstacken som en hiss. Varje skikt lägger till sin egen styrinformation i enlighet med protokollets logik. Det övre skiktets PDU utgör en av dataparametrarna.

Då meddelandet kommit fram till mottagaren påbörjar det sin hiss-liknande färd upp genom protokollstacken. I figur 14 beskrivs de olika skedena i skikt (N). Från skikt (N-1) kommer ett tjänsteprimitiv genom DOWN-gränssnittet. Funktionen `getDownParameters` avskiljer för skikt (N) specifik styrinformation och lagrar meddelandets tjänsteparametrar i virtualuppdragets lokala variabler. Då återstår den av partnerenheten sända PDU:n. Den dekodas av funktionen `getPeerParameters`, och även dess parametrar lagras i interna variabler. Protokollkontrollinformationen ges till automaten, som

därefter utför sin logik. Om den föreskriver att ett meddelande skall sändas uppåt till skikt (N+1) genom UP-gränssnittet anropas funktionen `putUpParameters`. Den konstruerar ett tjänsteprimitiv som tas emot av virtualuppdraget i det mottagande skiktet. På detta vis fortsätter meddelandet sin färd tills det kommit till mottagaren.



Figur 14. Meddelande till ett virtualuppdrag via DOWN-gränssnittet

4.3 Implementering av virtualuppdrag

Att implementera en protokollenhet i form av ett virtualuppdrag i CVOPS är ett flerstegsprojekt. Följande delprojekt bör utföras i ungefär denna ordning:

- definiera tjänste- och protokollgränssnitt
- beskriv den ändliga tillståndsautomaten
- skriv initialiseringsfunktioner i C
- skriv parameterfunktioner i C
- skriv kodnings- och dekodningsfunktioner i C
- skriv interaktiva testfunktioner i C

Protokollstackens arkitektur beskrivs i filen `struct.str`. Dessutom behövs det en associationsfil vid namn `vtaskassoc.c` i vilken virtualuppdragens typnamn binds till respektive associationsfunktioner. En associationsfunktion beskriver sambanden mellan ett virtualuppdrags olika gränssnitt och dess definitionsfiler.

Om protokolldefinitionen är skriven i språket ASN.1, vilket ofta är fallet med t.ex. tillämpningstjänsteelement, kan man generera programkoden till de två sista delprojekten automatiskt med hjälp av kompilatorn CASN.

4.3.1 Definitionsfiler

Nedanstående filer behövs vid implementering av ett virtualuppdrag. Ordet `vtask` ersätts med namnet på virtualuppdraget.

`vtask.if`	De av virtualuppdraget tillhandahållna tjänsterna.
`vtask.in`	Virtualuppdragets interna symboler.
`vtask.aut`	Virtualuppdragets tillståndsautomat.
`vtask.c`	Virtualuppdragets initialiserings-, parameter- och stödfunktioner.
`vtaskcod.c`	Virtualuppdragets kodnings- och dekodningsfunktioner.
`vtasktst.c`	Virtualuppdragets testfunktioner för interaktivt bruk.

Ifall protokollbeskrivningen är skriven i ASN.1 behövs även dessa två definitionsfiler:

`vtasks.asn`	ASN.1 beskrivning av de tillhandahållna tjänsterna.
`vtaskp.asn`	ASN.1 beskrivning av protokollet.

Eftersom protokollimplementeringen skrivs i C är det inget som hindrar att man t.ex.

samlar alla C-funktioner i en enda stor fil. Inom CVOPS-kulturen är det dock kutym att man spjälker upp programkoden i funktionella helheter. Det underlättar underhållsarbetet och tjänar dessutom som en enkel dokumentation.

4.3.2 Tjänstedefinitioner — `vtask.if`

Filen `vtask.if` definierar de tjänster som virtualuppdraget tillhandahåller. Eftersom det ovanliggande skiktet begagnar sig av dessa tjänster är denna fil gemensam för såväl det tjänsteerbjudande som för det tjänsteanvändande virtualuppdraget. Den innehåller namnen på tjänsteprimitiven och en beskrivning av det parameterblock i vilket datautbyte mellan skikten sker.

4.3.3 Interna symboler — `vtask.in`

I filen `vtask.in` definieras virtualuppdragets interna symboler: automatens tillstånd, PDU:n, timers, funktioner som anropas från automaten, lokala variabler och eventuella asn-variabler. Ifall CASN inte används definieras också ett protokollparameterblock i denna fil.

4.3.4 Tillståndsautomaten — `vtask.aut`

Tillståndsautomatena i CVOPS beskrivs med hjälp av regler och makrodefinitioner. En regel består av tre delar: tillstånd, insignal och handling. För varje tillstånd finns det en eller flera möjliga insignaler. Då automaten exekveras utförs den handling som definierats för det aktuella tillståndet och den aktuella insignalen.

En insignal kan vara t.ex. ett tjänsteprimitiv, en inkommande PDU eller ett timeoutmeddelande.

Regeln kan vara antingen specifik eller allmän. En specifik regel är en regel i vilken tillståndet och insignalen är bestämda. Handlingen utförs för exakt en kombination av tillstånd och insignal .

Kommentarer placeras mellan (* och *).

Nedanstående exempel är taget ur automaten för presentationsskiktet.

```
(* tillstånd      insignal        handling *)
   STAI0          up.p_conreq     {  initCRandRL()
                                     P_CP
                                     to(STAI1)
                                  }
```

Ovanstående regel tillämpas om presentationsskiktets virtualuppdrag befinner sig i tillståndet `STAI0` och från ACSE får en begäran att etablera kontakt med sin partnerenhet.

Följande handlingar utförs:

- den lokala makron `initCRandRL` anropas.
- åt partnerenheten sänds PDU:n `P_CP`.
- virtualuppdraget övergår i tillståndet `STAI1`.

En allmän regel är en regel som kan utföras endast om automaten inte innehåller en kombination av tillstånd och insignal som exakt motsvarar det faktiska förhållandet. Den beskrivs med tillståndet `GLOBAL` och/eller insignalen `OTHER`.

Följande regel utförs om ingen specifik regel passar in:

```
(* tillstånd      insignal        handling *)
   GLOBAL         OTHER           {  f_unexpInput()
                                     f_toSTAI0()
                                  }
```

Här anropas de lokala, i C skrivna funktionerna `f_unexpInput` och `f_toSTAI0`.

4.3.5 Initialiseringsfunktioner

Vid sidan om protokollets egentliga logik bör ett antal initialiseringsfunktioner också förverkligas. De har till uppgift att binda definitionerna till de rätta gränssnitten samt försätta virtualuppdraget i sitt utgångsläge. Följande initialiseringsfunktioner behövs:

`vtaskInit`	Virtualuppdragets initialiseringsfunktion. Här nollställs lokala variabler och reserveras minne för lokala datastrukturer. Ett drivuppdrag tillkännager här CVOPS sin s.k. händelsehanterare (event handler function).
`vtaskTypeInit`	Virtualuppdragstypens initialiseringsfunktion. Här binds basfunktionen, parameterfunktionerna och de i automaten anropade funktionerna till sina respekive pekare i typens datastruktur.
`vtaskAssoc`	I den här funktionen binds definitionsfilerna (`vtask.if`, `vtask.in` och `vtask.aut`) till sina gränssnitt.

4.3.6 Parameterfunktioner

Parameterfunktionernas uppgift är att förmedla data mellan ett parameterblock och en grupp lokala variabler. Parameterfunktionernas realisering beror alltså på parameterblockets struktur. Det finns fem typer av parameterfunktioner, av vilka varje gränssnitt har en egen uppsätting. Asterisken (*) i nedanstående lista ersätts med gränssnittets namn, t.ex. `getDownParameters`.

`get*Parameters`	Kopierar värden från meddelandet till virtualuppdraget.
`put*Parameters`	Kopierar värden från virtualuppdraget till meddelandet.
`ask*Parameters`	Frågar parametervärden av användaren.
`print*Parameters`	Skriver parametervärden på skärmen.
`free*Parameters`	Friger det minnesblock som tidigare reserverats för parametrar.

`put`- och `get`-funktionerna genomför parameterutbytet mellan virtualuppdragen. Terminalanvändaren kommunicerar med ett virtualuppdrag via `ask`- och `print`-funktionerna. De används bara under utvecklings- och testningsfasen.

Ifall tjänsterna och protokollet definierats på språket ASN.1 kan man låta kompilatorn CASN automatiskt generera ask/print-parameterfunktionerna och parameterblocken. Parameterblocken består då av asn-variabler.

4.3.7 Kodnings- och dekodningsfunktioner

Kodningsfunktionerna ansvarar för att meddelandet som sänds till partnerenheten är konstruerat i enlighet med standarden. De använder sig av värden i lokala variabler för att bygga upp ett meddelande i form av på varandra följande parametrar. Mottagarens dekodningsfunktioner läser dessa parametrar och lagrar dem i sina lokala variabler.

Kodnings- och dekodningsfunktionerna kan genereras automatiskt med hjälp av kompilatorn CASN ifall protokollets PDU:n är definierade med ASN.1.

4.3.8 Arkitekturbeskrivning

En protokollstack består av flera skiktenheter. Att implementera flera virtualuppdrag i CVOPS ger oss dock ingen stack förrän protokollstapelns struktur är fastslagen.

I filen `struct.str` finns namnet på alla virtualenheter som ingår i stacken. Dessutom framkommer det genom vilka gränssnitt följdenheterna är kopplade till varandra. Ett fragment ur en strukturfil kan se ut så här:

```
cmise    (trace)    up   :  cmif
                    down:   acse
                    resl:   roseb

acse                up   :  cmise
                    down:   pres

roseb               up   :  cmise
                    down:   pres

pres                up   :  acse
                    resl:   roseb
                    peer:   pres
                    down:   sessbb
```

Direktivet `trace` betyder att de meddelanden som kommer till virtualuppdraget `cmise` skall spåras på skärmen. Det är en mycket användbar egenskap i utvecklingsskedet.

4.4 Gränssnitt till externa system

CVOPS erbjuder programmeraren av datakommunikationsprotokoll en operativsystem-oberoende utvecklingsmiljö. En protokollimplementation behöver inte utvecklas i sin slutgiltiga arbetsmiljö, som kan vara ett modemkort eller en telefoncentral. Det kan man istället göra i en arbetsstation, varefter man porterar protokollstacken till slut-miljön.

För att underlätta förflyttningen av CVOPS från ett system till ett annat har man koncentrerat de operativsystemberoende aspekterna till några få moduler: *utils-modulen* och den tidigare nämnda inmatningsmodulen. En del av kommunikationsfunktionerna finns dessutom i drivuppdragen.

Utils-modulen tillhandahåller en mängd tjänster, men bara två av dem är systemspecifika och bör följaktligen återimplementeras i varje miljö, d.v.s. förvaltning av minne och läsande av systemklockan.

Reservering och frigivning av minne sker med hjälp av funktionerna `wmalloc()` och `wmfree()`. De tar i UNIX skepnaden av systemanropen `malloc(2)` och `free(2)`.

Tidsfunktionen ser i CVOPS ut på följande sätt: `wtime()`. Motsvarigheten i UNIX är denna gång ett anrop av biblioteksfunktionen `time(3)`.

För att protokollstacken skall kunna kommunicera med en tillämpningsprocess eller inbyggd kommunikationshårdvara krävs att den har tillgång till värdoperativsystemets faciliteter för interprocesskommunikation (IPC). Gränssnittet mellan operativsystemet och CVOPS placeras i huvudsak i drivrutinerna. I inmatningsmodulen finns stödfunktioner som drivrutinerna anropar. Inmatningsmodulen innehåller dessutom funktionen `waitInput` som, när de interna meddelandena tar slut, undersöker om det finns externa meddelanden att processera.

Då CVOPS' kärna levererar ett internt meddelande från ett vanligt virtualuppdrag åt ett drivuppdrag anropar CVOPS drivuppdragets basfunktion. Basfunktionen konverterar meddelandet till en tidigare fastslagen form och sänder det med hjälp av inmatningsmodulens funktioner till en yttre process. Drivuppdragets basfunktion ansvarar med andra ord för kommunikationen från protokollstacken till det omkringliggande systemet.

För att processera inkommande meddelanden har drivuppdraget dessutom en funktion för sk. händelsehantering. Då inmatningsmodulens funktion `waitInput` märker att det finns yttre meddelanden anropar den drivfunktionens händelsehanterare. Drivuppdraget läser då det yttre meddelandet och konverterar det till ett CVOPS-meddelande. Därefter sänder drivuppdraget meddelandet vidare i stacken.

4.5 Kompilatorn CASN

För att underlätta implementeringen av tillämpningsskiktets tjänsteelement, som alla är beskrivna med hjälp av språket ASN.1, har man vid Nokia Forskningscentral utvecklat CASN [CASN 1991a, 1991b], en kompilator för ASN.1. CASN kan användas för att utveckla såväl virtualuppdrag för CVOPS som ensamstående tillämpningar.

Utgående från ett elements tjänste- och protokollspecifikation i ASN.1 sammanställs sk. ASN.1 moduler som kompileras, varvid ASN.1 beskrivningarna omvandlas till typspecifikationer och virtualuppdragets asn-variabler bildas.

Med protokolldefinitionen som grund genererar CASN kodnings- och dekodningsfunktioner i C för protokolldataenheterna. Genom ett allmänt funktionsgränssnitt kommer man åt den specifika kodnings- eller dekodningsrutinen för den aktuella PDU:n.

Dessutom genererar CASN motsvarigheter till CVOPS' förfrågnings- och utskriftsfunktioner för interaktivt testningsbruk. Även dessa accesseras via allmänna funktionsgränssnitt. Med hjälp av testfunktionerna får man automatiskt ett rudimentärt användargränssnitt till protokollstacken.

5. Operativsystemet DNOS

DNOS [DNOS 1990] är ett operativsystem för modem. Det har utvecklats vid Nokia Datakommunikation (NDC). Dess syfte är att erbjuda en standardiserad, hårdvaruoberoende omgivning för system- och styrprogram och på så sätt befrämja framtagningen av portabel programvara. DNOS tillhandahåller ett väldefinierat gränssnitt till operativsystemets tjänster i form av systemanrop. Så gott som hela operativsystemet är skrivet i C.

Operativsystemet DNOS är, precis som UNIX, ett flerprocess-operativsystem. Det består av en systemprocess eller ett systemuppdrag (system task, scheduler) som handhar bl.a. fördelningen av körturen och ett godtyckligt antal tillämpningsprocesser. Processerna har fasta prioriteter.

Användarprocesserna begär tjänster av DNOS' kärna med hjälp av systemanrop. Det finns systemanrop för processförvaltning, synkronisation av processer och interprocesskommunikation (IPC).

DNOS är ett statiskt operativsystem. De flesta systemparametrar slås fast redan i kompileringsskedet. Bland dessa kan nämnas operativsystemets totala minnesrymd, antalet postlådor, semaforer och timers samt process-specifik data såsom namn, adress, prioritet, stackstorlek och ursprungstillstånd.

DNOS är inte heller något fristående operativsystem: systemprocessen, användarprocesserna och operativsystemets kärna kompileras och länkas till en enda modul. Alla processer är realiserade som funktioner i C.

NDC tillhandahöll en i Sun Microsystems' UNIX-variant SunOS [SunOS 1990] fungerande DNOS-emulator för diplomarbetets programutvecklingsskede. Den exekveras som ett UNIX-program. Precis som CVOPS kan också DNOS anses vara ett virtuellt operativsystem.

5.1 DNOS' kärna

DNOS' interna funktioner kretsar kring några centrala begrepp. De viktigaste datastrukturerna i DNOS är processen, postlådan, semaforen och operativsystemets listor.

Processen karakteriseras av bl.a. sin adress, sin prioritet och sin storlek samt det tillstånd den befinner sig i. Det finns fyra möjliga tillstånd. Processen kan vara *aktiv* (running), *redo* (ready), *suspenderad* (suspended) eller *sovande* (sleeping). Bara en process åt gången kan vara aktiv.

Processerna kommunicerar med varandra via postlådor. En postlåda är en datastruktur

till vilken en process kan sända ett meddelande. En postlåda kan läsas av vilken process som helst. Därför är det rekommendabelt att förse varje process med åtminstone en egen postlåda. Om en process försöker hämta ett meddelande från en tom postlåda avsäger den sig körturen och börjar sova. Då någon annan process sänder ett meddelande till lådan övergår den sovande processen i tillståndet redo. Man kan med andra ord synkronisera processer med hjälp av postlådor.

Ett meddelande består av tre fält: avsändare, typ av meddelande och ett datafält. Datafältets längd är endast 32 bitar. Därför överför man i allmänhet en pekare till ett datablock. Detta medför inga svårigheter eftersom alla processer har en gemensam minnesrymd.

Ändamålet med semaforer är att möjliggöra synkronisation av processer. En process kan tillkännage att den väntar på en semafor och suspendera sig själv. Då en annan process signalerar med semaforen återgår den suspenderade processen till tillståndet redo. I DNOS är det inte möjligt för flera processer att köa för en och samma semafor.

I DNOS' kärna finns det två dynamiska listor: en över processer i tillståndet redo (ready-list) och en över sovande processer (sleep-list). Processerna på listan ready-list är ordnade efter prioritet. Den första processen har den högsta prioriteten och är automatiskt aktiv. Då den aktiva processen avsäger sig körturen väljs nästa process på listan. För att en process med låg prioritet skall få köra måste med andra ord alla processer med högre prioritet antingen sova eller vara suspenderade.

Listan sleep-list är sorterad efter processernas sovtider: den process som har den kortaste sovtiden (som först har "sovit färdigt") ligger först. De övriga processernas sovtider är angivna som differensen mellan processens sovtid och den närmast föregående processens sovtid. Om det finns tre processer och de skall sova i 100 ms, 300 ms respektive 1000 ms så är deras sovtider angivna som 100 ms, 200 ms och 700 ms.

Vid uppstart reserverar DNOS `DMEM_SIZE` bytes primärminne. En del av det konsumeras direkt av systemet. Resten utgör dynamiskt minne som processerna kan reservera och frige under programmets gång genom systemanropen `Allocate()` och `Free()`. Systemanropet `Pool()` returnerar mängden fritt minne.

DNOS är ett icke-avbrytande operativsystem. En process i exekvering måste självmant deaktivera sig (avsäga sig körrättigheterna) antingen genom att anropa någon av systemfunktionerna `Pause()`, `Sleep()` eller `Suspend` eller genom att vänta på ett meddelande eller på en semaforsignal. Systemanropet `Pause()` avbryter den aktiva processen och placerar den på listan över körklara processer (ready-list) i enlighet med sin prioritet. Om flera processer har samma prioritet sätts den sist bland sina gelikar (pejus inter pares). Därefter ges körturen åt den process som nu är först på listan.

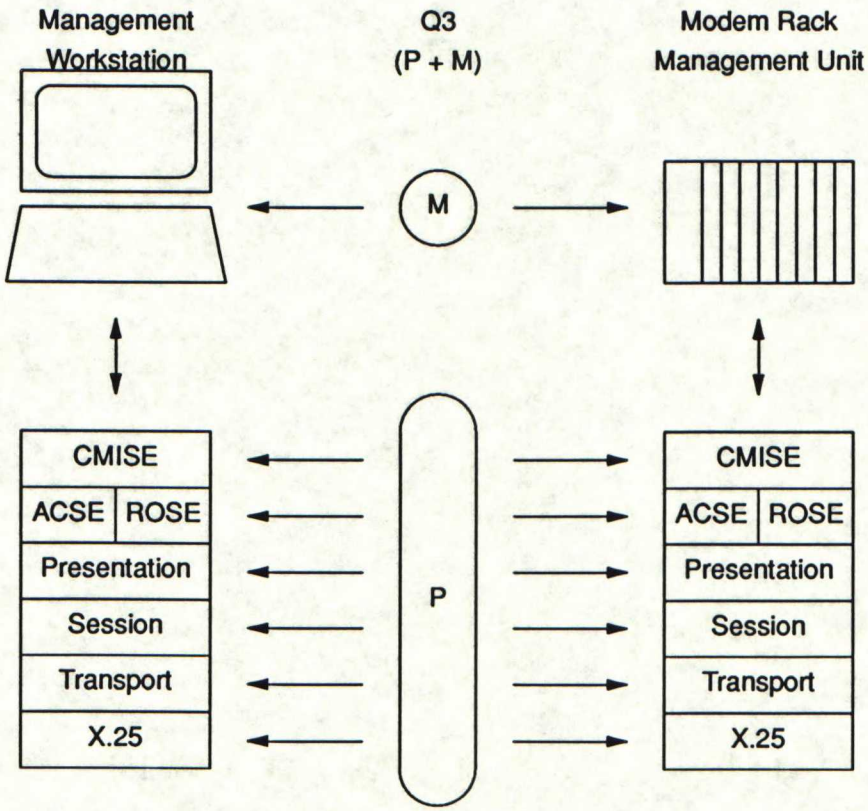
5.2 DNOS-emulatorn i UNIX

För att bespara författaren besväret med att utveckla program direkt på ett styrkort för modemstativ lät NDC göra en i UNIX fungerande emulator för DNOS. Emulatorn skrevs, precis som "riktiga" DNOS, nästan uteslutande i C. Endast kontextväxlingen (context switching) i samband med nytilldelning av körturen måste skrivas direkt i assembler. Kontextväxlingen manipulerar nämnligen processens stack, och den kommer man inte åt från ett vanligt program.

DNOS-emulatorns minnesförvaltning var ursprungligen dimensionerad för rätt blygsamma förhållanden: den maximala mängden dynamiskt minne var 64 kB. Eftersom en rejäl protokollstack i CVOPS kräver närmare 1 MB nödgades författaren göra ett smärre ingrepp i modulen för minnesförvaltning och omdefiniera några 16 bitars heltalsvariabler till 32 bitars dito.

6. Arbetets genomförande

Att övervaka och administrera kommunikationsaktiviteterna i ett modemstativ är en form av driftstöd. För att möjliggöra OSI-mässiga driftstödstillämpningar måste en Q3 (P+M) [CCITT 1991] protokollstack för driftstöd implementeras på modemstativets styrkort och i övervakningsarbetsstationen. I uttrycket P+M (se figur 15) representerar P (Protocols) de OSI-protokoll som används för att transportera driftstödsmeddelandena M (Messages) mellan modemstativet och övervakningsarbetsstationen. Kommunikationen mellan modemstativ och övervakande arbetsstation sker via X.25. Den interna kommunikationen i modemstativet mellan styrkort och modem sker på leverantörspecifikt sätt och berörs inte i detta diplomarbete. Protokollstaplarna på manager- och agentsida finns avbildade i figur 15.



Figur 15. Kommunikationen mellan modemstativ och arbetsstation

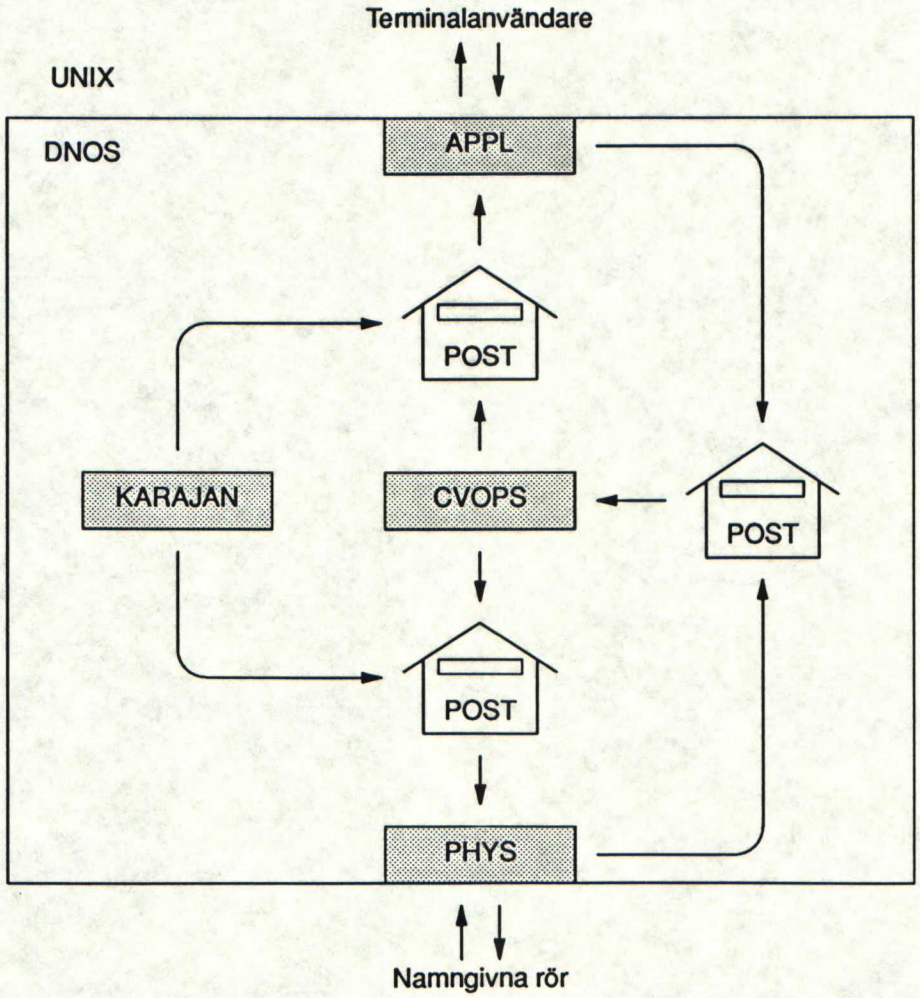
Styrkortet i modemstativet är försett med operativsystemet DNOS. Istället för att utveckla protokollstacken direkt på styrkortet har arbetet dock gjorts i en DNOS-emulator.

Ändamålet med diplomarbetet var, vid sidan om dess rent akademiska syfte, att anpassa och portera en i CVOPS implementerad Q3 protokollstack för driftstöd till

operativsystemet DNOS.

6.1 Omgivningen — program och apparatur

Hela protokollutvecklingen skedde i DNOS-emulatorn, som i sin tur utgjorde en process i SunOS (Sun Microsystems' variant av UNIX). I fortsättningen kommer DNOS-emulatorn att kallas enbart för DNOS och SunOS för UNIX. Figur 16 beskriver i detalj hur arbetet utfördes.



Figur 16. Q3 stacken i DNOS

De skuggade områdena betecknar processer i DNOS, som kommunicerar med varandra via postlådor. Processerna, som är fyra till antalet, är:

CVOPS	CVOPS innefattar, förutom protokollen i skikt 2 – 7, också de DNOS-specifika gränssnittuppdragen HDLCIF och CMIF i stackens bägge ändor samt inmatningsmodulen dnosinput. Dessutom utnyttjar CVOPS DNOS för minnesförvaltning och klocktjänster.
PHYS	Drivprocessen PHYS fungerar som gränssnitt mellan DNOS och UNIX. PHYS öppnar två namngivna rör för kommunikation med yttervärlden. CVOPS kan sända och mottaga meddelanden genom rören via PHYS.
APPL	Testprocessen APPL är ett menystyrt användargränssnitt till tillämpningsenheten CMISE.
KARAJAN	Drivprocessen KARAJAN har fått ta hand om alla småuppgifter som måste utföras under olika skeden av programmets livstid. Det kan närmast karakteriseras som ett slags allt-i-allo. Till dess viktigaste uppgifter hör initialiseringen av systemet vid uppstart och senare emulering av avbrott.

Hela DNOS-omgivningen, operativssystemet självt och alla dess processer, kompileras och länkas till ett enda program i UNIX. Istället för att använda en dedikerad övervakningsstation har under utvecklingsskedet två förekomster av programmet startats samtidigt och körts mot varandra.

Allt utvecklingsarbete skedde lokalt i en Sun SparcStation 2 arbetsstation utrustad med operativsystemet SunOS 4.1.1. Som C-kompilator användes den som ingår i standarddistributionen av SunOS. DNOS versionen var 2.0.

6.2 OSI-stacken

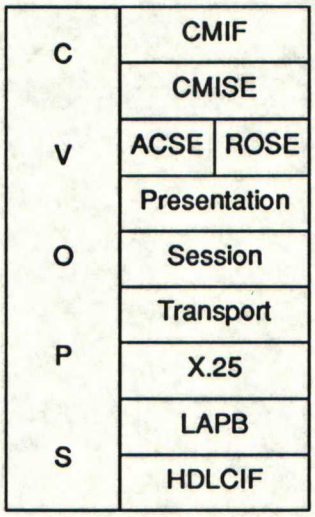
CVOPS protokollstacken i DNOS består av 3 logiska helheter. Protokollskikten 2 – 7 (LAPB – CMISE) utgör själva kärnan. De ingår i alla implementationer av CCITT:s protokollsvit B1 för driftstöd oberoende av utvecklingsomgivning.

Gränssnittuppdragen HDLCIF och CMIF ombesörjer kommunikationen mellan CVOPS och dess omgivning. HLDCIF har till uppgift att konvertera och överföra meddelanden mellan skiktenheten LAPB och drivprocessen PHYS. CMIF utför samma uppgift mellan skiktenheten CMISE och testprocessen APPL. De använder sig bägge av DNOS' postlådekommunikation.

Inmatningsmodulen dnosinput, som tillhör CVOPS' kärna, tillhandahåller direkta

funktionsanrop för kommunikation i DNOS. Via dnosinput väntar CVOPS' kärna också på externa meddelanden.

De tre huvudgrupperna och deras ömsesidiga förhållanden finns illustrerade i figur 17.



Figur 17. Detaljerad bild av Q3 protokollstacken i DNOS

7. Anpassning av CVOPS till DNOS

För att anpassa CVOPS till DNOS måste, som tidigare nämnts, funktionerna för minnesförvaltning och klocktjänster i CVOPS' utils-modul modifieras något. Reservering av minne åstadkoms i DNOS med hjälp av systemanropet `Allocate()` som returnerar en pekare till ett minnesområde av önskad storlek. Ifall minnet inte räcker till returnerar funktionen värdet noll.

Frigivning av ett tidigare reserverat minnesblock sker i DNOS via systemanropet `Free()`. Om minnesblocket inte tidigare reserverats med `Allocate()` returnerar funktionen värdet ERROR.

Minnesförvaltning i CVOPS anpassades till DNOS genom att ändra CVOPS' funktioner för reservering och frigivning av fysiskt minne, `wmalloc` och `wmfree()`, till att i sin tur anropa `Allocate()` respektive `Free()`.

CVOPS' klocktjänst baserar sig på sk. väggklockstid, medan DNOS i normala fall har timers och därför kan erbjuda endast sk. äggklockstid. Att erbjuda väggklockstjänster med hjälp av en äggklocka är förvisso inte omöjligt, men det ansågs i varje fall onödigt. Istället försågs modemstativets styrkort med en hårdvaruklocka och såväl DNOS som DNOS-emulatorn utvidgades med ett systemanrop för att läsa klockan. Detta anrop baserades för enkelhetens skull på motsvarande systemanrop i UNIX och gavs också samma namn: `time()`.

7.1 Inmatningsmodulen dnosinput

Alla yttre meddelanden till CVOPS kommer via en och samma postlåda. För enkelhetens skull heter den också CVOPS. På basen av meddelandets avsändare och en global datastruktur kallad `devinfo` kan CVOPS' kärna dirigera meddelandet till rätt gränssnittuppgift. Vektorn `devinfo` ser ut så här:

```
typedef struct {
    unsigned int flag;          /* Entry free or in use? */
    BYTE_TYPE from;             /* DNOS task sending msg */
    VTASK *vtask;               /* If-vtask handling msg */
    void (*event_handler)();    /* Handles incoming msg */
} DEVICE_INFO;

static DEVICE_INFO devinfo[MAX_DEVICES];
```

Då ett meddelande från DNOS-processen `from` kommit till CVOPS' postlåda söker

inmatningsmodulen rätt på det aktuella vektorelement och verifierar att virtualuppdraget `vtask` accepterar meddelanden från `from`. Därefter anropar inmatningsmodulen den händelsehanterare (`event_handler()`) vars pekare finns i det aktuella vektorelementet. Händelsehanteraren läser sedan meddelandet och omvandlar det till ett internt CVOPS-meddelande.

Varje virtualuppdrag som kan ta emot meddelanden från en yttre process, dvs. HDLCIF och CMIF, placerar sina egna data i vektorn `devinfo` genom att vid uppstarten anropa inmatningsmodulens funktion `enterDevice(vtask, from, handler)`. Funktionen `enterDevice()` söker fram det första lediga elementet i vektorn och tilldelar det sina argument.

För att gränssnittuppdraget HDLCIF skall kunna ta emot meddelanden från PHYS anropar dess initialiseringsfunktion för enhetsvirtualuppdraget `hdlcifVtaskInit()` funktionen `enterDevice(HDLCIF, PHYS, handlePhysBox)`.

Då initialiseringsfasen är undanstökad börjar CVOPS processera sina interna meddelanden. Om det inte finns flera interna meddelanden kallar CVOPS' huvudprogram på funktionen `waitInput()`. Den undersöker först om någon timer har gått ut. Om så är fallet bildar CVOPS ett timer-meddelande och börjar bearbeta det. I annat fall beräknar CVOPS tiden tills nästa timer går ut (`delta`) och anropar funktionen `event_io(delta)`.

Funktionen `event_io()` väntar antingen tills det kommit ett meddelande i postlådan CVOPS, i vilket fall den anropar mottagaruppdragets händelsehanterare, eller tills tiden `delta` gått ut och CVOPS kan bearbeta ett timeout-meddelande.

7.2 Allt-i-allo processen KARAJAN

En väsentlig skillnad mellan DNOS-emulatorn i UNIX och det "riktiga" operativsystemet som finns på modemstativets styrkort är att styrkortets DNOS har direkt tillgång till den underliggande hårdvaran. Då kortets kommunikationskontroller får ett meddelande utifrån placerar den det i en intern buffert och genererar ett avbrott. Avbrottshanteraren hämtar meddelandet från bufferten och placerar det i processens PHYS postlåda. Meddelandet träder så att säga "av sig själv" in i DNOS' domäner.

DNOS-emulatorn är en vanlig process i UNIX, och kan följaktligen inte omprogrammera avbrottshanterarna efter behag. Därför har KARAJAN fått ikläda sig rollen som avbrottshanterare.

KARAJAN är den process som har den lägsta prioriteten. Den blir aktiv endast då CVOPS har processerat alla interna meddelanden och kallat på `waitInput()` som i sin tur anropat `event_io()` för att hämta ett meddelande från postlådan CVOPS. Om

postlådan är tom sover processen tills den får ett meddelande. Då aktiveras KARAJAN.

I kompileringsskedet har KARAJAN bibringats information om genom vilka kanaler ett meddelande kan komma från UNIX-omgivningen till DNOS. KARAJAN använder sig av UNIX systemanropet `select()` för att övervaka dessa kanalers filbeskrivare (file descriptor). Anropet återvänder då det detekterar I/O-aktivitet och returnerar den filbeskrivare via vilken data kan läsas. På basen av fildeskriptorn och en global datastruktur vid namn `inputinfo` lokaliserar KARAJAN den DNOS-process som meddelandet är avsett för. Till denna drivprocess sänder KARAJAN ett meddelande av typen `INTERRUPT`. Den suspenderade drivprocessen övergår i tillståndet redo, ges körturen och läser meddelandet från UNIX' kommunikationskanal, omvandlar det till ett DNOS-meddelande och skickar det till CVOPS' postlåda. På detta sätt kan DNOS-omgivningen asynkront ta emot meddelanden utifrån.

Förutom den ovan beskrivna uppgiften som avbrottsemulator är KARAJAN också den process som drar i gång CVOPS och drivprocesserna i början. KARAJAN ansvarar för att processens PHYS' namngivna rör öppnas och för att systemanropet `select()` får de rätta parametrarna.

8. Gränssnittuppdraget HDLCIF

Syftet med gränssnittuppdraget HDLCIF är att isolera protokollstackens nedersta skikt från det omkringliggande operativsystemet DNOS. HDLCIF konverterar CVOPS-meddelanden från virtualuppdraget LAPB till DNOS-meddelanden och sänder dem till DNOS-processen PHYS. Beroende på om detta utspelar sig i DNOS-emulatorn i UNIX eller på modemstativets styrkort vidarebefordar PHYS meddelandet antingen till en annan UNIX-process via namngivna rör eller till styrkortets kommunikationshårdvara.

Även den motsatta uppgiften, att ta emot ett DNOS-meddelande från PHYS, omvandla det till ett CVOPS-meddelande och sända det till skiktenheten LAPB, ingår i HDLCIF:s plikter.

Gränssnittuppdraget HDLCIF är ett virtualuppdrag i CVOPS, men det har ingen protokollfunktion. Därför behöver det heller ingen tillståndsautomat. HDLCIF innefattar, förutom basfunktionen som ombesörjer kommunikation från CVOPS till DNOS, också den tidigare nämnda händelsehanteraren. Den förmedlar meddelanden från DNOS till CVOPS.

Virtualuppdraget LAPB kräver inget annat av HDLCIF än överföring och mottagande av färdigt kodade meddelanden (sk. ramar). Eftersom LAPB [ISO 1986] kanaliserar flera länkförbindelser på en enda fysisk förbindelse behöver HDLCIF inga förbindelseuppdrag. HDLCIF behöver ej heller tillstånd, interna variabler eller timers. HDLCIF har ett enda gränssnitt, UP, och erbjuder bara två tjänsteprimitiv: P-DATA.request och P-DATA.indication.

Den av HDLCIF tillhandahållna tjänsten P-DATA har ett enkelt parameterblock som består av enbart en bytevektor. Parameterfunktionerna `getUpParameters()` och `putUpParameters()` kopierar datablocket från CVOPS-meddelandet till en lokal bytevektor och vice versa.

Gränssnittsuppdragets egentliga begäran realiseras i form av två funktioner: basfunktionen `hdlcifBody()` och händelsehanteraren `handlePhysBox()`. Basfunktionen tar emot en uppgift från virtualuppdraget LAPB i form av tjänsteprimitivet P-DATA.request, omvandlar dataparametern till ett DNOS-meddelande och sänder det till processen PHYS' postlåda. Händelsehanteraren gör det omvända, den accepterar ett DNOS-meddelande från PHYS, omvandlar det till en tjänstedataparameter och levererar tjänsteprimitivet P-DATA.indication till LAPB.

Vid uppstart registrerar HDLCIF sig och sin händelsehanterare i inmatningsmodulens datastruktur `devinfo` genom att i virtualuppdragets initialiseringsfunktion `hdlcifVtaskInit()` anropa inmatningsmodulens funktion `enterDevice()`.

8.1 Samspelet mellan HDLCIF och PHYS

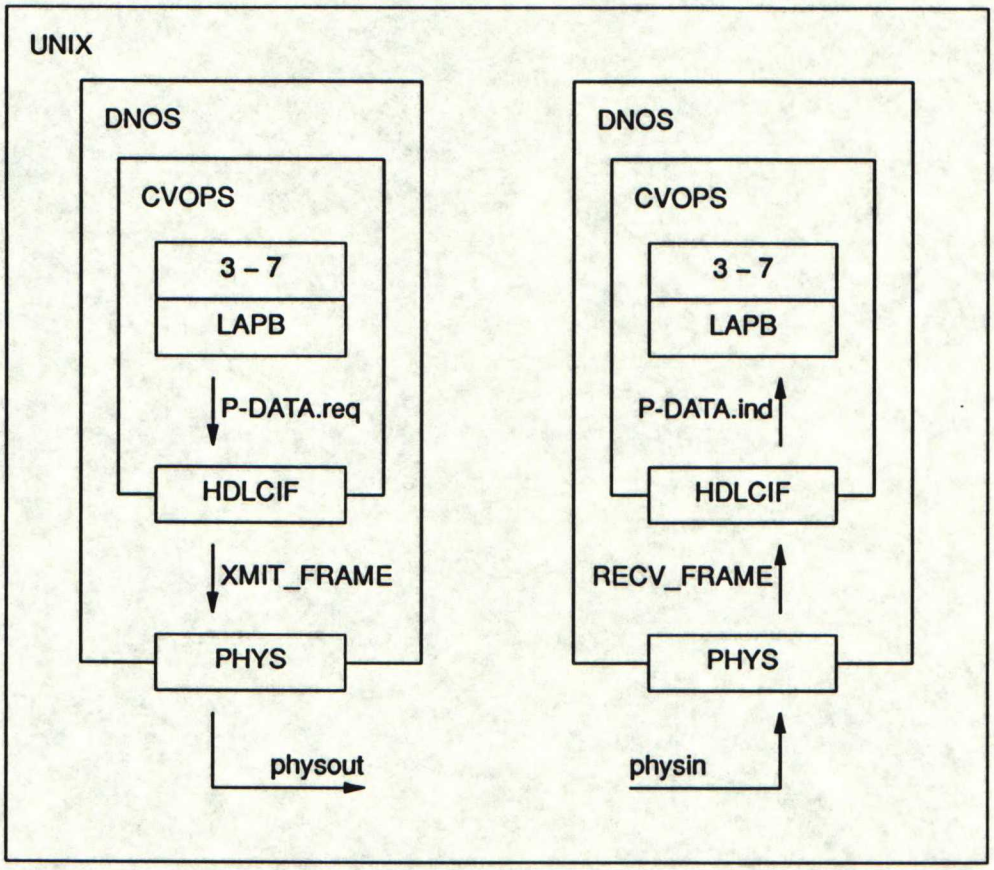
Ett DNOS-meddelande är, som tidigare nämnts, en C `struct` med tre fält: `from`, `type` och `msg`. Det sistnämnda fältet är en pekare till en bytevektor, som i detta fall består av en två bytes lång längdindikator och själva meddelandet.

Då HDLCIF får ett CVOPS-meddelandet från LAPB undersöker funktionen `hdlcifBody()` längden på tjänsteprimitivets dataparameter och reserverar minne för såväl meddelandet som längdindikatorn direkt från DNOS. I denna bytevektor fyller basfunktionen först längdindikatorn och sedan det serialiserade meddelandet. Då detta gjorts sänder den ett DNOS-meddelandet av typen `XMIT_FRAME` till PHYS.

Orsaken till att `hdlcifBody` reserverade minne direkt från DNOS och inte via CVOPS är att det för meddelandet reserverade minnesutrymmet friges i processen PHYS. PHYS har naturligtvis inte tillgång till CVOPS' minnesförvaltning. Om utrymme gång på gång reserveras i CVOPS men aldrig friges tar primärminnet så småningom slut. Detta förfarande, att ett uppdrag i CVOPS reserverar minne direkt från DNOS, används i bägge gränssnittuppdragen.

DNOS-processen PHYS läser det inkommna meddelandet, som består av en längdindikator och ett datafält. Den omkodar längdindikatorn till ett fyra bytes långt hexadecimalt tal och skriver det i en lokal buffert. Därefter kopierar PHYS datafältet till bufferten efter längdindikatorn. Med hjälp av UNIX systemanropet `write()` skriver PHYS bufferten i det namngivan röret `physout` och friger sedan DNOS-meddelandets datablock.

Då PHYS får ett `INTERRUPT`-meddelande av processen KARAJAN läser den en bytevektor från röret `physin` medelst UNIX systemanropet `read()`. Vektorn, som består av en längdindikator och en dataparameter, omvandlas till ett DNOS-meddelande av typen `RECV_FRAME` som sänds till processen CVOPS. Inmatningshanteraren `dnosinput` får tag på meddelandet och, på basen av tabellen `devinfo`, anropar funktionen `handlePhysBox` med meddelandet som argument. Händelsehanteraren hämtar meddelandet från postlådan, konverterar det till ett CVOPS-meddelande och sänder det genom gränssnittet UP som ett tjänsteprimitiv P-DATA.indication till LAPB. Därefter friger den det minnesblock som PHYS reserverat. En schematisk skiss över meddelandet kan ses i figur 18.



Figur 18. Meddelandets färd från LAPB i CVOPS via DNOS och UNIX till LAPB

9. Gränssnittuppdraget CMIF

Gränssnittuppdraget CMIF påminner i mångt och mycket om sin kollega HDLCIF. Bägge har till uppgift att verka som gränssnitt mellan protokollstacken i CVOPS och det omkringliggande operativsystemet DNOS. CMIF ligger ovanför tillämpningselementet CMISE och förmedlar meddelanden mellan det och driftstödsagenten APPL. Även CMIF har bara ett CVOPS-gränssnitt: DOWN.

Virtualuppdraget CMISE erbjuder driftstödsapplikationen ett flertal tjänster och stöder dessutom flera samtidiga förbindelser. CMIF får naturligtvis inte inskränka på service-nivån utan även det måste hantera flera förbindelser och flera tjänster. Till en följd av detta är CMIF betydligt mera komplext än HDLCIF.

Virtualuppdraget CMIF har varken interna tillstånd eller tillståndsautomat. Däremot har det samma uppsättning asn-variabler som i CMISE betjänar UP-gränssnittet. Eftersom de omkringliggande protokollenheterna i skikt 6 och 7 är utvecklade med hjälp av såväl CVOPS som CASN har parameter- och kodningsfunktionerna genererats automatiskt. Då två med CVOPS och CASN implementerade virtualuppdrag sänder CVOPS-meddelanden åt varandra består dataparametern inte av en enda lång bytevektor utan av en tabell med pekare till i minnet utspridda dataenheter av typen ASNHOME. En sådan distribuerad dataparameter kan inte utan vidare utväxlas mellan CMIF och tillämpningen, utan dataparametern måste först *serialiseras* och efter mottagningen *deserialiseras*. Det har visat sig praktiskt att i dylika fall använda BER, som trots att det inte producerar den mest kompakta serialiseringen har den fördelen att det ingår i CASN.

CMIF:s basfunktion `cmifBody()` och händelsehanterare `handleApplBox()` är betydligt mera komplicerade än de motsvarande i HDLCIF. Då CMISE utför tjänsterna M-INITIALIZE, M-TERMINATE eller M-ABORT måste CMIF initialisera respektive nollställa datastrukturer för förvaltning av förbindelser.

Vid uppstart registrerar virtualuppdraget CMIF sig och sin händelsehanterare i inmatningsmodulens datastruktur `devinfo` genom att i virtualuppdragets initialiseringsfunktion `cmifVtaskInit()` anropa inmatningsmodulens funktion `enterDevice()`.

9.1 Samspelet mellan CMIF och APPL

De tjänster CMIF tillhandahåller kan samtidigt utnyttjas av flera tillämpningsprocesser. Dessutom kan varje tillämpningsprocess ha flera parallella förbindelser. CMIF måste kunna identifiera från vilken process ett meddelande kommer och till vilken förbindelse det hör. För att möjliggöra detta förvaltar CMIF en tabell vid namn `Connections`, i

vilken varje öppen förbindelse finns beskriven. Ett element i denna tabell ser ut så här:

```
#define MAXCONN 20

typedef struct {
    BYTE_TYPE state;            /* Entry free/in use? */
    int appl;                   /* Appl owning conn */
    int connId;                 /* Conn ID in appl */
    int ownCepId;               /* Conn ID in cmif */
    int cmiseCepId;             /* Conn ID in cmise */
} CONNECTION;

CONNECTION Connections[MAXCONN];
```

Fältet `state` ger vid handen huruvida vektorelementet är i bruk eller inte. Dess möjliga värden är `ALLOCATED` och `UNALLOCATED`. Fältet `appl` indikerar till vilken tillämpningsprocess den aktuella förbindelsen hör. Alla tillämpningsprocesser har en egen postlåda med samma namn som processen. Då man önskar kommunicera med processen `appl` sänder man alltså ett meddelande till postlådan `appl`. En tillämpningsprocess' olika förbindelser skiljs åt med hjälp av fältet `connId`.

Fälten `ownCepId` och `cmiseCepId` representerar förbindelsens ändpunktsidentifikation i virtualuppdragen CMIF respektive CMISE. Med hjälp av dem kan en förbindelse följas genom protokollstacken upp via CMISE till CMIF, där man via vektorn `Connections` kommer åt paret (`appl, connId`) som entydigt identifierar tillämpnigsprocessen och vilken förbindelse det är frågan om.

En förbindelse kunde identifieras som t.ex. tillämpningsprocessens APPL3 (`appl=3`) andra förbindelse (`connId=2`) med förbindelseändpunkten nummer 4 i CMIF (`ownCepId=4`) och nummer 1 i CMISE (`cmiseCepId=1`).

Meddelandehuvudet `MSG_HEADER` är en annan datastruktur av vikt. Alla DNOS-meddelanden som utväxlas mellan CMIF och tillämpningarna består av ett huvud som innehåller styrinformation och en kropp som utgörs av tjänstedataenheten. Meddelandehuvudet är definierat enligt följande:

```
typedef struct {
    short connId;               /* Conn ID of msg */
    short prim;                 /* Primitive */
    long len;                   /* Length of msg */
} MSG_HEADER;
```

Ett meddelande från CMIF till en tillämpning adresseras till tillämpningens postlåda. För att specificera vilken förbindelse det är frågan om finns dess nummer i `connId`. För att undvika att CMIF och tillämpningen allokerar samma nummer för olika

förbindelser genererar CMIF alltid ett jämnt tal och tillämpningen ett udda tal. CMIF levererar flera olika tjänsteprimitiv mellan tillämpning och protokollstack. Fältet `prim` indikerar vilket primitiv som transporteras i det aktuella meddelandet. I längdindikatorn `len` finns DNOS-meddelandets längd.

Då CMIF tar emot ett CVOPS-meddelande av typen `RECV_CMISDU` från DNOS-processen APPL kopierar händelsehanteraren `handleApplBox()` först meddelandets huvud till en lokal variabel och sedan, då meddelandets längd är känt, själva tjänstedataenheten till en lokal bytevektor. Om tjänsteprimitivet visar sig vara M-INITIALIZE.request måste CMIF finna ett ledigt element i vektorn `Connections` och initialisera det. Fältet `cmiseCepId` lämnas än så länge tomt; det fylls i först när partnerenhetens bekräftelse M-INITIALIZE-confirmation levereras av CMISE.

Därefter reserverar CMIF minne för ett CVOPS-meddelande att sända till CMISE, fyller i dess fält med aktuell information och deserialiserar DNOS-meddelandets dataparameter för att placera dess värden i sina asn-variabler som utgör CVOPS-meddelandets parameterblock. Till sist sänder `handleApplBox()` meddelandet till CMISE genom att anropa funktionen `sendMessage()`.

Om CMIF får ett tjänsteprimitiv från CMISE anropas dess basfunktion `cmifBody()`. Ifall primitivet är M-INITIALIZE.indication reserverar funktionen det första tomma elementet i vektorn `Connections` och fyller i dess fält. Därefter serialiserar basfunktionen dataparametern, kopierar över den i en buffert som reserverats direkt från DNOS och konstruerar ett huvud för meddelandet. Avslutningsvis frigör `cmifBody()` den serialiserade dataparameterns minnesblock och sänder ett DNOS-meddelande av typen `XMIT_CMISDU` till tillämpningen.

9.2 APPL — en rudimentär testapplikation

Den enda skillnaden mellan de meddelanden som utväxlas mellan å ena sidan CMISE och CMIF och å andra sidan CMIF och APPL är att det sistnämnda meddelandet är serialiserat och och bär med sig ett huvud med styrinformation. Processen APPL använder sig nämnligen av samma parameterfunktioner som virtualuppdraget CMISE, fastän den inte är något virtualuppdrag i CVOPS. Eftersom alla processer i DNOS har en gemensam minnesrymd kan APPL komma åt variabler och funktioner i övriga processer. Vid uppstart postar CMIF pekaren till sin s.k. *entrydescriptor*-tabell till APPL, och genom denna pekare kommer APPL åt sina parameter-, kodnings- och testfunktioner.

APPL är en menystyrd tillämpning för testning av Q3-protokollstacken. Huvudmenyn erbjuder terminalanvändaren tre möjligheter: att sända ett CMISE-tjänsteprimitiv till CMISE via CMIF, att ta emot ett CMISE-tjänsteprimitiv från CMISE eller att avsluta programmet.

Om terminalanvändaren väljer att sända kommer en ny meny i vilken APPL listar alla CMISE-tjänsteprimitiv och frågar vilket primitiv som skall sändas och till vilken förbindelse det hör. Därefter anropar APPL CASN-funktionen `askAsnParameters()` som frågar terminalanvändaren vilka värden primitivets parametrar skall ha. Till sist förser tillämpningen meddelandet med ett huvud och sänder det till CMIF. För att ta emot ett meddelande väljer terminalanvändaren mottagning i huvudmenyn. APPL hämtar meddelandet ur sin postlåda och anropar CASN-funktionen `printAsnParameters()` som skriver primitivet och dess parametrar på skärmen.

10. Sammanfattning

I detta diplomarbete implementerades en sk. Q3-protokollstack för modemstativ i operativsystemet DNOS. Arbetet bestod av att portera protokollutvecklingsverktyget CVOPS till operativsystemet DNOS och att sammanställa en OSI-konform protokollstack av färdiga virtualuppdrag i CVOPS. För att anpassa stacken till värdoperativsystemet gjordes gränssnittuppdragen HDLCIF och CMIF. En menystyrd testapplikation, APPL, skrevs så att protokollstackens korrekthet kunde verifieras. Arbetet gjordes i en DNOS-emulator i UNIX. Kommunikationen mellan DNOS och UNIX realiserades i form av två DNOS-processer, PHYS och KARAJAN.

Protokollutvecklingsverktyget CVOPS' kärna och de i CVOPS utvecklade virtualuppdragen utgjorde tillsammans en process i DNOS. Processen PHYS, som i modemstativet skulle driva kommunikationshårdvaran, fungerade som gränssnitt mellan DNOS och UNIX genom att vidarebefordra meddelanden från en postlåda i DNOS till namngivna rör i UNIX. För att övervaka och upptäcka inkommande meddelanden fick processen KARAJAN till uppdrag att emulera en del av de avbrottsfunktioner modemstativets hårdvara i allmänhet erbjuder. Den interaktiva testprocessen APPL erbjuder användaren samma tjänster som tillämpningselementet CMISE.

Den färdiga helheten, ett i UNIX kompilerat och statiskt länkat program bestående av operativsystemet DNOS med processerna CVOPS, KARAJAN, PHYS och APPL, upptar ca 730 kB skivutrymme. Vid körning reserverar programmet ungefär 130 kB dynamiskt minne. Alla samband som etableras till andra protokollstackar behöver dessutom vidpass 10 kB per samband.

Testtillämpningen APPL var av ytterst blygsam karaktär. En "riktig" driftstödsapplikation skulle vara betydligt större. Driftstöd i enlighet med OSI kräver med andra ord mycket av apparaturen både i form av minnesmängd och räknekapacitet.

Bibliografi

[Abramowicz 1989] H. Abramowicz & A. Lindberg: "OSI för telekommunikationstillämpningar". *Ericsson Review,* nr. 1, 1989.

[Ahtiainen 1990] A. Ahtiainen, J. Keskinen, J. Simolin, K. Tarpila & I. Turunen: "Protocol Software Engineering Tools for Implementation of a General Purpose OSI Stack" in *Computer Networking,* ed. L. Csaba, Szentivanyi & K. Tarnay. Elsevier Science Publishers B.V (North Holland), Amsterdam, Nederländerna, 1990.

[CASN 1991a] A. Ahtiainen, S. Kesti & M. Turunen: "CASN Compiler Reference Manual". Nokia Research Center, Esbo, 1991.

[CASN 1991b] A. Ahtiainen, S. Kesti & M. Turunen: "CASN User's Manual". Nokia Research Center, Esbo, 1991.

[CCITT 1991] Principles for a Telecommunications Management Network. Comité Consultatif Internationale Télégraphique et Téléphonique, 1991. Recommendation M.30.

[CVOPS 1990] J. Harju & J. Koivisto: "CVOPS User's Guide". Telcommunications Laboratory of VTT, Esbo, 1990.

[DNOS 1990] DNOS dokumentation och källkod. Nokia Data Communications, 1990.

[ISO 1984] Information Technology — Open Systems Interconnection — Basic Reference Model. International Organization for Standardization and International Electrotechnical Committee, 1984. International Standard 7498.

[ISO 1989a] Information Technology — Open Systems Interconnection — Application Layer Structure. International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 9545.

[ISO 1987a] Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8824.

[ISO 1987b] Information Technology — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). International Organization for Standardization and International Electrotechnical Committee, 1987. International Standard 8825.

[ISO 1988a] Information Technology — Open Systems Interconnection — Service Definition for the Association Control Service Element. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8649.

[ISO 1988b] Information Technology — Open Systems Interconnection — Protocol Specification for the Association Control Service Element. International Organization for Standardization and International Electrotechnical Committee, 1988. International Standard 8650.

[ISO 1989b] Information Technology — Open Systems Interconnection — Remote Operations Part 1: Model, Notion and Service Definition. International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 9072-1.

[ISO 1989c] Information Technology — Open Systems Interconnection — Remote Operations Part 2: Protocol Specification. International Organization for Standardization and International Electrotechnical Committee, 1989. International Standard 9072-2.

[ISO 1990a] Information Technology — Open Systems Interconnection — Service Definition for the Common Management Information Service Element. International Organization for Standardization and International Electrotechnical Committee, 1990. International Standard 9595.

[ISO 1990b] Information Technology — Open Systems Interconnection — Protocol Specification for the Common Management Information Service Element. International Organization for Standardization and International Electrotechnical Committee, 1990. International Standard 9596.

[ISO 1986] Information Technology — Data Communications — High-level data link control procedures — Description for the X.25 LAPB-compatible DTE-data link procedure. International Organization for Standardization and International Electrotechnical Committee, 1986. International

Standard 7776.

[Kärkkäinen 1991] E. Kärkkäinen: "Avoimen tietoverkon hallinta". Villmanstrands tekniska högskolan, Villmanstrand, 1991. Diplomarbete.

[Kurronen 1991] K. Kurronen: "Lyhytsanomakeskuksen liittäminen GSM-matkapuhelinverkkoon". Tekniska högskolan, Esbo, 1991. Diplomarbete.

[NDC 1990] ECM 4896M TRELLIS Yleisseloste. Nokia Data Communications, 1990.

[OSI/NM Forum 1989] OSI/Network Management Forum. OSI/NM Forum, 1989-1991.

[Rose 1990] M.T. Rose: "A Practical Perspective on OSI". Prentice Hall, 1990. ISBN 0-13-643016-3.

[SunOS 1990] SunOS Reference Manual. Sun Microsystems, 1990.